B.Tech - Project
on

Autonomous Car Project - Automated Driving System Design and Implementation

(Project code: PI-11)

Submitted by:
Akash Mahajan - 2016ME20748
Dhruv Talwar - 2016ME10670

Supervisor:
Prof. Sunil Jha
Department of Mechanical Engineering

June 2020

Acknowledgment

We would like to express our heartfelt gratitude to our project supervisor - Professor Sunil Jha, for his continuous encouragement, ceaseless engagement, and useful critiques for this project. It has been an honor to work under his mentorship. We would also like to thank Mr. Souvick Chatterjee from Mathworks, who was always proactive in addressing any of our concerns related to the MATLAB Toolboxes.

Abstract

The following thesis discusses a section of the parent project, focusing on the development of an Autonomous Controller for a driverless car. The primary aim is to design an Automated Driving System with lane assist and forward-collision warning system which could be run successfully on IIT Delhi roads.

Designed algorithms, used preexisting libraries on MATLAB, and then also integrated it with ROS for live communication. Gazebo simulation systems, video sources, and even live video feed was used for testing some of the algorithms and the qualitative comparison of the designed models. The main target is to exploit the capabilities of MATLAB and use its inbuilt libraries, models, and algorithms to achieve the desired output and control signals.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background

In the last century, the mechanics, control, and efficiency of a car have constantly and considerably improved, but a lot still has to be done to achieve the peak performance that can be achieved by a personal automobile vehicle. The automobile industry is expected to undergo some massive technological changes in the near future. The concept of intelligent vehicles or self-driving cars has become some of the most challenging problems of AI. A driverless car has the ability to generate advanced control output based on a variety of sensor data using cameras, radars, GPS, Odometry. It has to combine the need for optimal and smooth processing with the safety of its passengers and the environment.

In the past few years, considerable research and investment have gone into driverless cars. The field has become more formalized and has become the center of innovation for ventures around the world like Waymo, Tesla. Self-driving has divided into various levels according to the increasing complexity of its autonomous nature.

Level 0: No Automation – The control system is completely handled by the driver. The car can only output some specific sensor information.
Level 1: Hands-on – The autonomous system does not have an independent existence, and it acts as a support to the driver in steering or acceleration.
Level 2: Hands off – The driver assist system is able to control steering and acceleration independently but requires continuous monitoring by the driver and must be ready for any manual intervention.
Level 3: Eyes off – The automated system can also handle emergency situations. But the driver is notified about an emerging situation that might need manual handling.
Level 4: Mind off – The car can be completely autonomous, but in a certain range, so the driver has to be in the car.
Level 5: Full Automation – No driver needed. Completely autonomous

The self-driving system implemented in the paper automates the control of the car but under constant supervision of a driver (Level 2). Software systems like MATLAB and ROS have been used to achieve the same.

1.2 Overview

For this undergraduate thesis, the aim is to design a vision based assist system capable of controlling the e2o car autonomously, keeping the stereo camera as the global sensor. The idea is to use the technology which is developed by Mathworks and mold it according to the required needs. Initially, the focus was on understanding and working on the MATLAB vision toolbox, but later, the approach was diversified. The algorithms were tested on simulation and real-world videos. ROS-MATLAB framework was also explored to integrate the results and outputs to control the car.

Chapter 2: Literature survey

The field of autonomous vehicles is witnessing rapid developments, it is now required that upcoming vehicles should have the necessary capabilities of lane driving.
Khalifa et al. presented a novel vision system that used cameras that were looking outward from inside of the car. This algorithm takes in images from the camera and detects the lanes. The lanes are extracted using Hough transform through a pair of parabolas which are fitted to the edges of the lane[33]. Seo et al. presented the use of a monocular camera to analyses a sequence of images to generate information about the steering of the vehicle and to keep it inside the lanes of the road. This is used extensively for lane driving.[34]

Moreover, driving support system has now become one of the essential features in an autonomous vehicle to ensure the safety of the driver and its surroundings. Aytekin et al. proposed a vehicle detection and tracking which was based on monochrome images taken by monocular camera. They relied on lighting conditions but proposed ways to incorporate shadows in the algorithm[35]
For pedestrian tracking Gilmore et al. developed a low computation pedestrian detection system using IR sensors on a MATLAB platform[36]. As pedestrian detection is different from standard detection as on-road accidents with pedestrians can cause loss of life. Zhang et al. discussed the feasibility of Fast- RCNN and other neural networks for detecting pedestrians. They finally proposed a simple but effective threshold for pedestrian detection.[37]

MATLAB is an opensource tool that is excellent for prototyping and simulating robotics control[38][39]. The platform that will be used for the vehicle control would be ROS, thus any software which would be selected should be ROS compatible. Earlier ROS was simply used with either python or C++ nodes thus for every functionality algorithms and codes were developed from scratch. Earlier it was also challenging to integrate ROS and MATLAB. The need for such a unification of Both raised when researchers all over the world wished to utilize the plethora of functions that MATLAB has to offer. But since the development of the Robotic System Toolbox in MATLAB[40] now a simple and effective bridge between them can be formed.

Motivated by the MATLAB capabilities, in this thesis a forward collision warning system and a vision-based lane following steering control system is developed. Both the systems are facilitated by using the pedestrian, vehicle and lane detection capabilities of MATLAB. All the above said tasks are completed using a single monocular camera.

Chapter 3: Project Objectives and Workplan

3.1 Problem Definition/ Motivation

Team DLive at the Indian Institute of Technology Delhi is aiming to develop a fully autonomous vehicle that can provide a safe driverless travel. Using cameras and LIDAR, various models and algorithms can be made to add perception which can be integrated with the control system to produce vehicle outputs like steering, acceleration and braking. The driverless car vehicle should also be responsive to its immediate dynamic environment to take any emergency measures. Since it's a developing field, the complexity of the problem can be further increased by adding capability to drive in dim lights, identify bumps and even traffic signs.

3.2 Objectives of the work

The objective is to be able to run a car driverlessly at Level 2 stage of automation.

The project goals are:

- Exploring MATLAB for automated driving system.
- Implementing motion planning and control on the video output of a monocular camera using visual perception.
- Develop a forward-collision warning system using vehicle and pedestrian tracking.
- Form a Bridge between ROS and MATLAB platforms for testing on Mahindra e2o.

3.3 Methodology

Continuous vision feedback was used to achieve the goal was controlling the car driverlessly. Initially, a literature survey was done to study some of the existing platforms for Designing Autonomous Driving Systems like Nvidia Isaac and MATLAB. It was decided to proceed with MATLAB to develop perception systems and algorithms.

On successfully running the vision models of lane detection, pedestrian, and vehicle detection, they were used for developing the control mechanism of the car. Some more models and algorithms were explored for path planning, trajectory generation, including some miscellaneous functions like traffic signal detection. The algorithms can be improved and filtered after practically executing them in the car.

Chapter 4: Motion Planning using Visual Perception

4.1 Linear Lane Boundary Detection

For this Autonomous Driving Feature, a monocular camera system was incorporated. The reading returned by the monocular camera sensor can be used to issue lane departure warnings, collision warning. It is also used to implement an emergency braking system in section 5. Using this system, lane boundaries were detected as straight lines and markers were also generated in vehicle coordinates (APPENDIX A).

In this algorithm, after breaking down the video stream into a flow of frames, the following significant techniques were used for lane detection –

1. Gaussian blur: The frame was initially passed through a Gaussian smoothening function[1] with a default standard deviation of 0.5 to remove the minute details and noise in the image. MATLAB function imgaussfilt3() was deployed for the same. It applies the Gaussian function (using equation 4.1) on each pixel as the center and that pixel's new value is set equal to the weighted average of its neighbor pixels. Since the Gaussian function achieves its peak value at the center, the pixel itself has the highest weight and its neighbor add a reducing weight with increasing distance till 3σ, after which the influence becomes negligible. This generates a blurred filtered image.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{- (4.1)}$$

2. Color Masking: Masking the original image (Fig 4.1) for white color to detect the lane lines perfectly. An iterative process was used to set optimal threshold values of Hue, Saturation, and value (HSV format). For the video mentioned in the report, for all the 3 parameters, a range of 170-250 was found to be optimal. After setting this range, using Boolean tools the inliers were set to 1 and others to 0, generating a black and white image, as seen in Fig 4.2 .The filtered image still contained a lot of noise / extra white pigment at this stage.

Fig 4.1 Original Road Image

Fig 4.2 White Mask applied

3. Canny Edge Detector: Implemented the Canny edge detection algorithm[2] using MATLAB edge() function which finds the first derivative in the X and Y directions (equation 4.2) on the masked image and calculates the slope of the derivative (equation 4.3). Any gradient in the pixel values is identified and marked using the slope values, thus giving highlighted edges.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \quad \text{- (4.2)}$$

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x) \quad \text{- (4.3)}$$

A MATLAB bewareaopen() function was then used on edge detected image to remove small connected areas with less than a threshold number of positive pixels. By trying various values, a threshold of 50 connected pixels was chosen that removes small spots on the road but also leaves lane lines untouched.

4. Region of Interest: An appropriate quadrilateral region is defined separately for computing the right and left lanes in different frames and also removes the majority of external noise. Using roipoly() function on 720x1280 image frame, the 2 quadilaterals were defined and plotted by joining (200,300), (700,300), (600,1000), (0,1000) and (900,100), (1100,100), (1500,1000), (1200,1000) as seen in Fig 4.3 and Fig 4.4 respectively . The ROI is particularly defined for the following sample and varies for different videos.

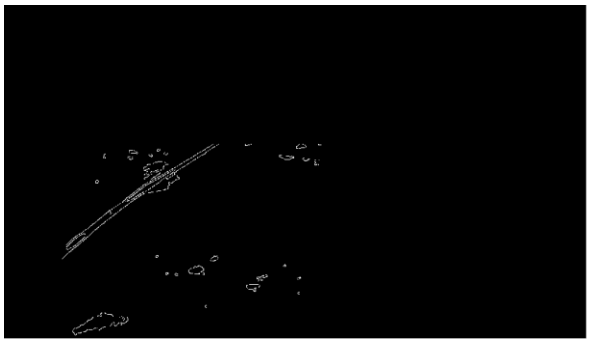

Fig 4.3 ROI of Left Lane

Fig 4.4 ROI of Right Lane

5. Hough Transform Line detection: On the binary edge detected images, for each highiteed (x,y), a sinusoid is plotted in the polar plane using the equation 4.4 -

$$\rho = x\cos(\theta) + y\sin(\theta) \quad \text{- (4.4)}$$

The points on the graph with the greatest number of intersections show high confidence that a line exists with those (ρ, θ).This was achieved using the houghpeaks() function which outputs the best fitting hough lines[3]. The maximum number of peaks to detect for this function was set to 3. The road lanes consist of 2 edge lines and there is a possibility of any noise which may also be detected as a line, so 3 peaks give a high assurity that the lane edges are not missed.

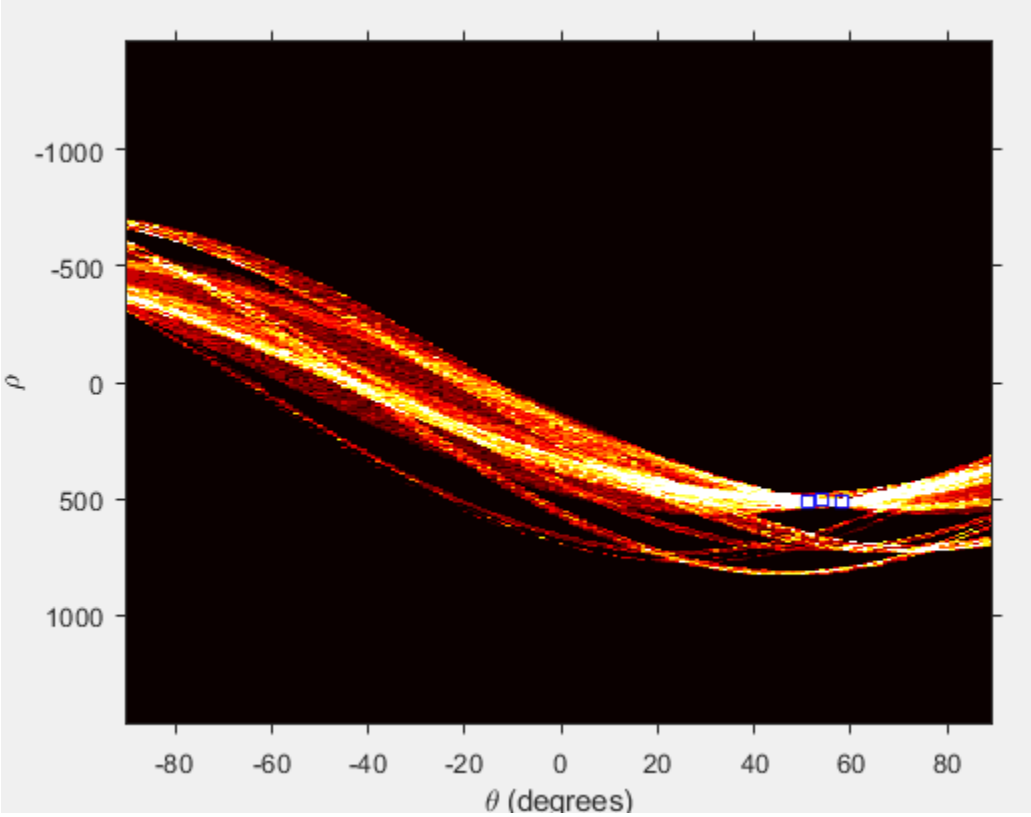

Fig 4.5 Hough plot for left lane

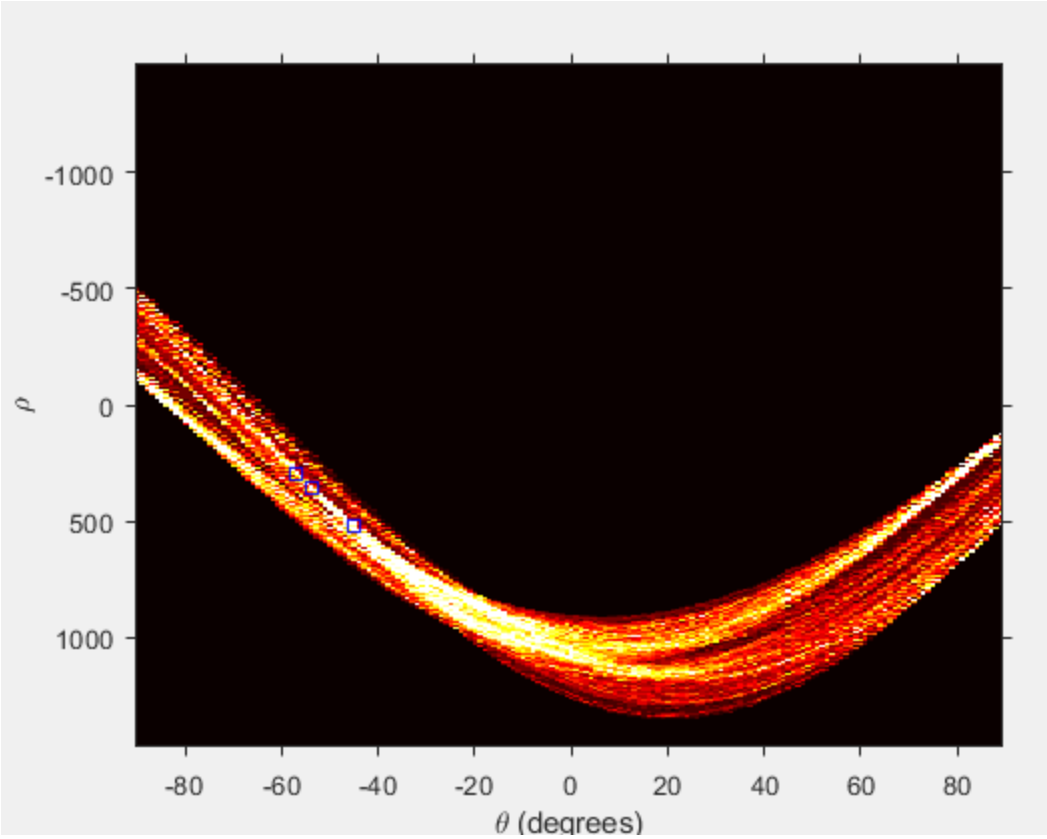

Fig 4.6 Hough plot for right lane

Using houghlines() function, the peaks (marked by blue boxes in Fig 4.5 and Fig 4.6) were then converted to parametric line plots by using the (ρ, θ) values and the best fitting lines for both the lanes (using an average of endpoints of the Hough lines) were extrapolated and then plotted on the main filtered image as shown in Fig 4.7.

Fig 4.7 Linear Lane Detection

4.2 Monocular Camera Calibration

For all the demonstrations in Section 4.3, Chapter 5 and Chapter 7, the HP Wide Vision HD camera is used. Since this method uses a conversion between pixel and vehicle coordinates, camera calibration was needed[4].

Calibration is a technique used to improve the quality of images captured the camera by correcting for lens distortion or to measure of the dimensions in world units.
The intrinsic parameters of a camera such as focal length, and optical center helps in converting pixel distance to real world distances. Extrinsic parameters include the height at which the camera is mounted or the pitch at which the camera is set.

For camera calibration the MATLAB the computer vision toolbox app namely the camera calibrator was used. More than 30 images of a checkerboard calibration. A checkerboard is used because its regular pattern makes it easy to detect automatically. The size of the checkerboard square was 24mm, and the arrangement was a 9X7 one. Once all the parameters are added to the app, it then automatically detects the pattern in the images provided (Fig 4.8(a)(b)(c)). Once the calibration is done by the app the results can then be seen either in the workspace (Fig 4.9) or as a MATLAB file.

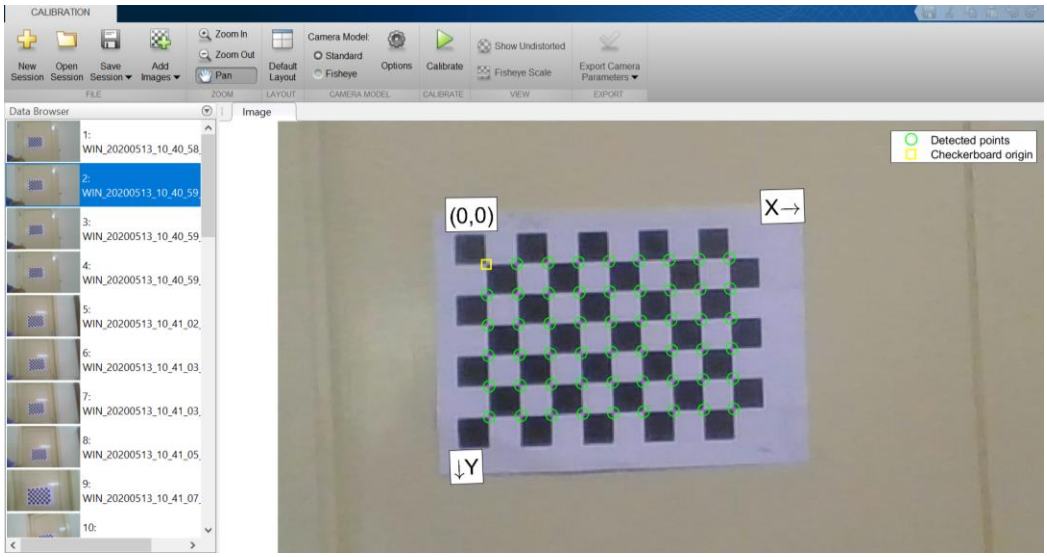

Fig 4.8 (a) Detection of the checkerboard pattern

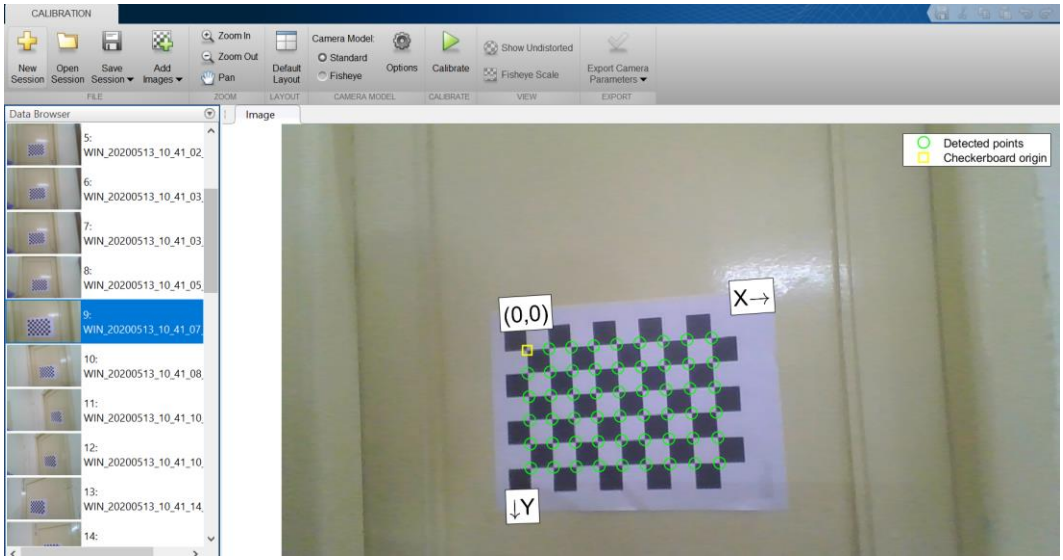

Fig 4.8 (b) Detection of the checkerboard pattern

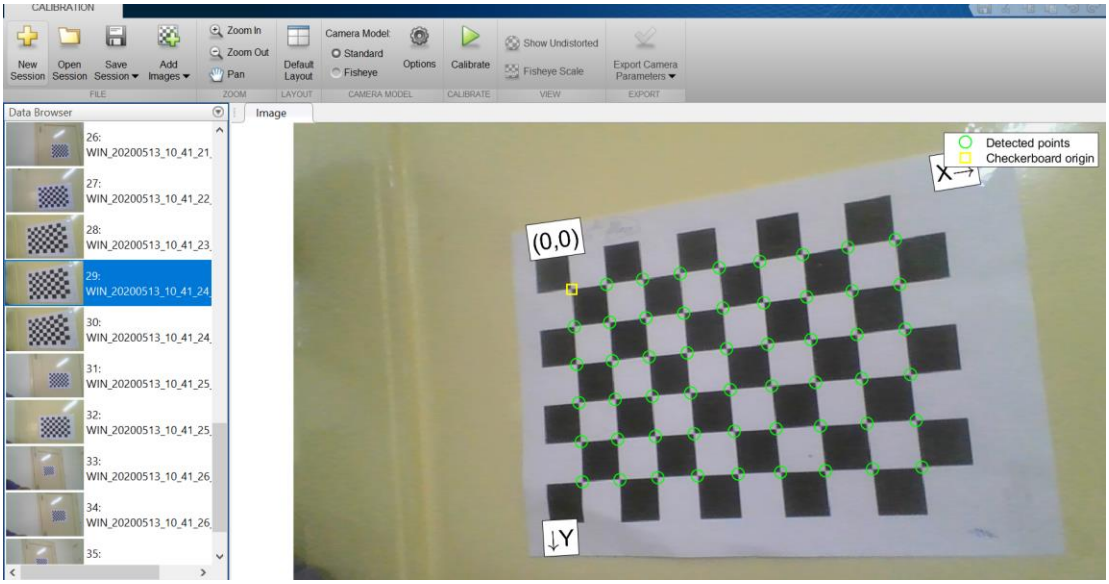

Fig 4.8 (c) Detection of the checkerboard pattern from various angles

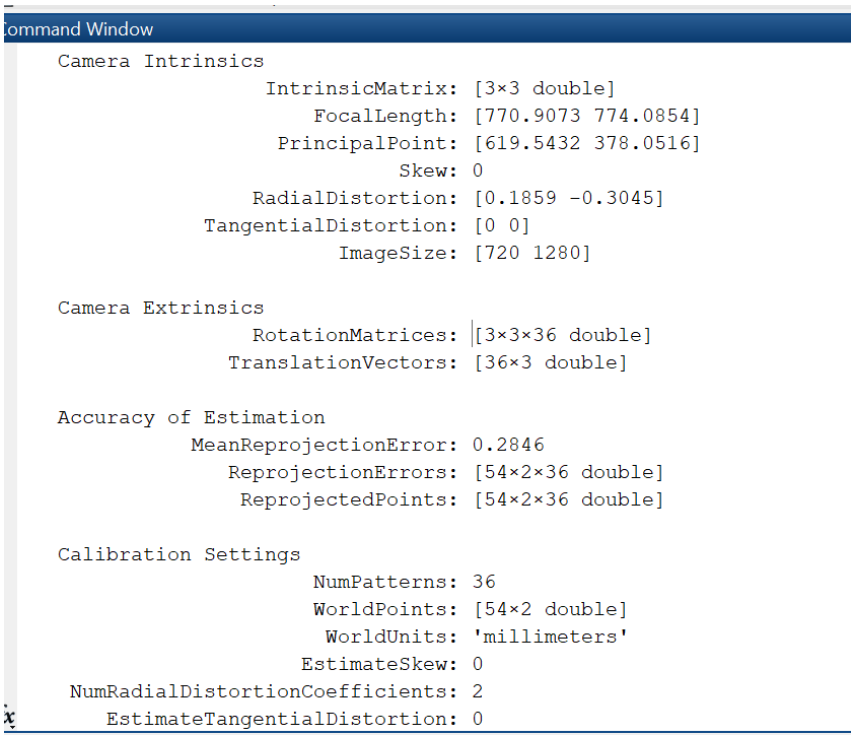

Fig 4.9 Calibrated Camera results

The calibration was done 5 times, and the final values which were needed were the average of the 5 calibration results. The calibration results gave the intrinsic values that were needed. Each time any world to pixel or vice versa conversion was needed, the camera intrinsic values were to be used for this. For the HP Wide Vision HD camera, which was used, the values came out to be

Table I Camera parameters Used

Focal Point	[771.6633 773.6966]
Optical Center	[622.8061 379.3357]
Image Size	[720, 1080]

The other parameter that was set was the height above which the camera was located while recording. This height was measured using a measuring tape.

4.3 Parabolic Lane Boundary Detection

In this algorithm, parabolic lane boundary model[5] was used as opposed to the simple linear one (APPENDIX B). A parabolic curve with equation $y=Ax^2+Bx+C$ is tried to be fitted on lane lines using suitable values of parameters [A,B,C]. The video was broken into a input stream of images and then operated upon.

Fig 4.10 Original Road Image

The input image frame (Fig 4.10) was initially converted to Birds Eye View[10] to ensure that the lane markers are parallel and also of uniform thickness. To implement the same, a birdsEyeView object was made using the camera extrinsic and intrinsic and also specifying the transformation in vehicle coordinates. The following area (defined in meters) was chosen since it covered a considerable road area and was also largely unaffected by the sudden camera pitch angle changes (less stable since camera was held in hand).

```
distAheadOfSensor = 30;
spaceToleftSide    = 10;
spaceTorightSide   = 10;
bottomOffset       = 1;
```

This BEV image (Fig 4.11) was then converted to a binary image using the rgb2gray(), and then a MATLAB inbuilt function – segmentLaneMarkerRidge()[6] was deployed to separate the lane marker candidate pixels from the road surface and gives quite a reliable binary mask of the lanes with minimum noise (Fig 4.12). It tries to eliminate some of the possible candidate detections using inputs like –

1) A custom ROI = 1 meter to the left, 1 meter to the right, 3 meters in front of sensor
2) Approximate width of the lane markers = 25cm

Fig 4.11 Birds Eye Image of Road

Fig 4.12 Binary Image with candidate lanes

Then, the lane candidate points were extracted from the BEV image using the same BirdsEyeView object and transformed to world coordinates to get a better sense of curvature without any perspective distortion. On these arrays of points, parabolicLaneBounadary()[7] function was deployed which uses a RANSAC algorithm to fit the most appropriate curve (parabola) and also find the polynomial coefficients[8],[9]. The algorithm carries out the following steps -

1. A small sample set is randomly selected from input data – hypothetical inliers, and the best fit model (2^{nd} degree equation) and its parameters are computed using these data points.
2. In the second step, the rest of the data is then fitted using the curve.
3. The algorithm checks the total number of inlier data points, decided by a predefined error threshold.
4. The above process is repeated iteratively and the fit that has the most number of inliers is selected.

The output lane boundary model consists of properties of the best fit curves (Fig 4.13) -

```
boundaries =

  1×2 parabolicLaneBoundary array with properties:

    Parameters
    BoundaryType
    Strength
    XExtent
    Width
```

Fig 4.13 Boundary model properties

1. Parameters – Real valued vector [A B C] of the best fit curve $y = Ax^2+Bx+C$.
2. Boundary Type – Type of boundary i.e. solid, dashed etc.
3. Strength – Ratio of the number of unique x coordinates on the curve to the length of the curve projected along x-axis (using XExtent).
4. XExtent – Real valued vector [Xmin Xmax] with minimum and maximum x coordinate values of the curve.

Some heuristics like the minimum length of the lane marker, lane strength were further applied to make the algorithm more robust. The detections with a very small value of A (<0.003) were also rejected since very small A value means a very high curvature. A function validateBoundaryFcn() was made to implement the same (APPENDIX B). Then, these lane boundaries are plotted on the bird's eye view (Fig 4.14) and the original image (Fig 4.15) using insertLaneBoundary() which takes as input the sensor or camera data and the boundary model.

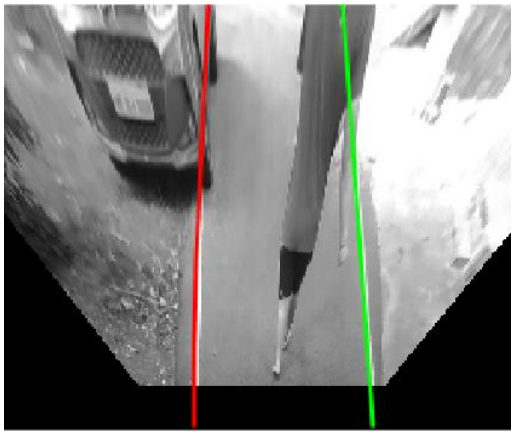

Fig 4.14 Lane Detection in Birds Eye view of Road

Fig 4.15 Lanes on actual Image

4.4 Comparative Analysis

To test both the algorithms, they were tuned and run on a video with much external noise in the form of sunlight, shadows, white environment, and also sudden movements of the camera. On proper calibration, it is seen that though the results from Linear lane detection (Fig 4.16) were not bad, but Parabolic lane detection (Fig 4.17) gave very consistent lanes without sudden jumps and is also able to map the curves better. This is becomes necessary for computing correct and reliable steering angles.

Fig 4.16 Linear Lane detection

Fig 4.17 Parabolic Lane detection

4.5 Lane following - Steering

In the Linear lane detection algorithm, for finding the steering angle (APPENDIX A) the following approach was used –

1. Define a point at the center of the camera frame represented by a red marker.
2. Now define another deviation point (represented by the blue marker) with y coordinate at center of the frame height and the x coordinate computed using the mean of the x coordinates of the lane boundaries.
3. Using these 2 points as 2 of the vertices the triangle, define a 3rd vertex at the bottom of the frame with x coordinates as half of width of the frame.
4. The deviation angle or the angle covered by the red and blue markers on the 3rd vertex can be found by -

   Deviation Angle = $\tan^{-1}$((x of blue marker – x of red marker) / (height/2) - (4.5)
5. This base deviation angle can be scaled according to the steering sensitivity to get the desired steering angles.

Fig 4.18 Steering Linear Lane detection

As can be verified by the Fig 4.18 that on a right turn, the technique defined is able to detect it and generate a value which prompts the car to turn right.

To find the steering angle in Parabolic lane detection (Fig 4.19), it was decided to use the same process as followed in the Linear lane detection algorithm. A novel technique

was used in which a function steer_fun() (APPENDIX B) was designed, that generates a black frame of the given image dimensions and plots the final lane boundaries on it in white color to return a pseudo-white-masked image. On this image, edge detection (MATLAB function edge()) and hough line transformations (MATLAB function hough()) as defined in the Linear lane model were applied to get the best-fit lane slopes and coordinates. Then the steering angle was calculated using the process defined at the start of Section 4.4.

Fig 4.19 Steering Parabolic Lane detection

In both figures above the angle of steering is measured in degrees. The absolute value of the steering value is not of importance. As depending on the road curve slope, the sensitivity of the steering algorithm could be tweaked after a few trial runs with the actual car.

Certain thresholds were also added like the steering is only active when a certain angle (4 degrees defined for Linear lane model) is achieved, because turning of a car based on very light deviations and noise is not desirable. Similarly, very high steering values which are not expected on a conventional road turn might be due to a wrong lane detection and therefore a cap of 15 degrees was put on it (for Linear lane model).

Chapter 5: Forward Collision Warning System

5.1 Working of Pedestrian and Vehicle Detection Codes

Caltech University[11] had already developed a pretrained vehicle and pedestrian detector. This detector can be imported into the MATLAB workspace and then be used on live video feed for detection purposes. The name of the dataset is the “Caltech Pedestrian Dataset”. The dataset has more than 200,000 frames and has more than 350,00 annotations. The trained detector is very easy to import in MATLAB and thus can be used for further development. Training a new detector with new data is not feasible due to the time constraints.
MATLAB offers choices in the model architecture to be used.

a. Aggregate Channel Features
b. Faster RCNN
c. Yolov2

ACF[12][13][15] was chosen as it took the least amount of time and computation power as compared to the other two. Also, while running different models on the same video ACF gave quite good results while maintaining decent frame rates. The workflow for both the vehicle detector and the pedestrian detector is the same.

Aggregate Channel detector is a common object detection method used in computer vision. A group of channels would be export from an input image, then some features will be obtained by pixels in the channel around a bounding box. Channels are defined as blocks of various pixel values.[21][22]

Forward Collision using Vehicles and Pedestrians tracking

Basic Initialization

In this part first the basic necessary objects are initialized. This included setting up the camera, the CNN based detector and setting the video on which the detection algorithm is to be run.

Firstly, the camera is calibrated according to the procedure followed in Chapter 4 Section 4.2. After calibration the camera intrinsic parameters are added to the MATLAB workspace. The camera parameters used are given in TABLE I. Then the camera mounting height which was earlier measured is defined. Once all this is done, everything can be combined to initialize the sensor object for this code.

Vehicle and Pedestrian Tracking (Forward collision system)

The vehicleDetectorACF() and the peopleDetectorACF() (APPENDIX C) are used to initialize vehicle and pedestrian detectors respectively. Both work on the Aggregate Channel Feature model. Then the average width of vehicles and pedestrians are defined. Only a bounding box which comes in this range might be considered as an image. Thus, two variables are defined vehicle_Width and ped_Width. (APPENDIX C). For the vehicle the average width defined is between 1.5 and 2.5 meters, And for the pedestrian the width defined is between 0 and 1 meters.

Using the configureDetectorMonoCamera() function both the pedestrian and vehicle detector is then configured with the sensor values which was initialized earlier and with the pedestrian and vehicle width value set.

The pedestrian_setup_tracker() and the vehicle_setup_tracker() are used to create a multiobjecttracker() which will be used to track multiple objects using the Kalman filter[14]. For both, pedestrian and vehicle tracking, the function has the following arguments

1. Filter Initialization: The apparent motion and measurement models is defined for the filter. In this example, constant velocity motion is assumed.
2. Distance Threshold: A distance is defined as how far detections can fall from the track.
3. Deletion Threshold: A number is defined as the number of times the track was not assigned to a detection was missed in the last S step before its deletion for detections
4. Confirmation Threshold: This is the parameter which is used for confirming a new track. A new track will be initialized with a new unassigned detection.

Both the pedestrian_setup_tracker() and the vehicle_setup_tracker() give an output of a a multiobjecttracker() and a positionselector.

Both the pedestrian_setup_tracker() and the vehicle_setup_tracker() call another function within them which is then used to initialize the Kalman Filter. Both these functions call pedestrian_init_Bbox_filter() and the vehicle_init_Bbox_filter(). These functions are called to initialize a Kalman filter which will be used in the pedestrian_setup_tracker() and the vehicle_setup_tracker() function. In the KF initializing functions the model which will be used is defined. As time variant transitions are ignored for this example, the constant velocity model is selected. Because the constant velocity model is used the acceleration is neglected.

The video reader is then loaded to play the already recorded video. The setup can be changed to take input from a live video. A loop is generated to break the video into frames and the loop shall be active till the frames in the video are finished. In the loop the frame counters are updated, and next frame is given to the embedded functions with each iteration.

In the loop first the pedestrian_detect_objects() and vehicle_detect_objects() are called. A detector(pedestrian and vehicle), the current frame and the current time stamp are the inputs to this function. This function uses both the detectors set, to return the results into an object which is then used by the multiobjectTracker(). In both functions the respective detector is run on the image or the frame, which is provided, and a list of bounding boxes is returned. The bounding box is of type [x,y,w,h]. These bounding boxes define where the respective detection is in the frame and help in detecting people or cars in the frame. The output of these functions are pedest_detections and vehicle_detections. Using this list of bounding boxes, the tracks are returned for the current time step. This is done for both the pedestrian and the vehicle detections.

After this the pedestrian_remove_noisy_tracks() and the vehicle_remove_noisy_tracks() functions are called. This function removes the noisy tracks. A track or a detection might be considered noisy if the predicted bounding box is quite small. This implies that the object is too far away. This function is run for both the pedestrians and vehicles. The function then extracts the object position from the detections. The boxes which are too small are classified as invalid. And are not taken into consideration.

Finally, the pedestrian_insert_box() and the vehicle_insert_box() function is used to add the annotations to the frame. The function extracts the positions of the bounding boxes. Then these relative positions of the boxes are converted into the vehicle or the pedestrian coordinates using the sensor object which was initialized in the beginning. For each detection a variable known as “labels” is defined which contains the vehicle ID and the X and Y coordinate of the vehicle or the pedestrian. Inside this function only the value of alpha (α) is defined.

Safe threshold distance Alpha (α) - This is the minimum distance between the ego vehicle(The vehicle on which the camera is mounted) and any nearby object, after the actual distance becomes lower than the α value the vehicle will get a message start to apply brakes. The brakes might not be instantaneous and can be a function of the distance alpha. The alpha value is different for both pedestrians and vehicles. While it might be acceptable to come very close to a vehicle but, protecting human life is a necessity. Therefore, the value taken for humans is much greater than the value for vehicles. The more the value of α the less severity of the brakes applied and vis versa. The value of alpha is defined here, and if the distance goes below this value, a message shall be sent to the ROS control to apply brakes

The alpha value for both vehicles and pedestrians are defined. For the pedestrians the value is higher than vehicles as explained above. Then from the label’s variable the coordinates of the nearest vehicle or pedestrian is extracted. Fig 5.1 shows the layout of the X and Y coordinate as printed out by the code.

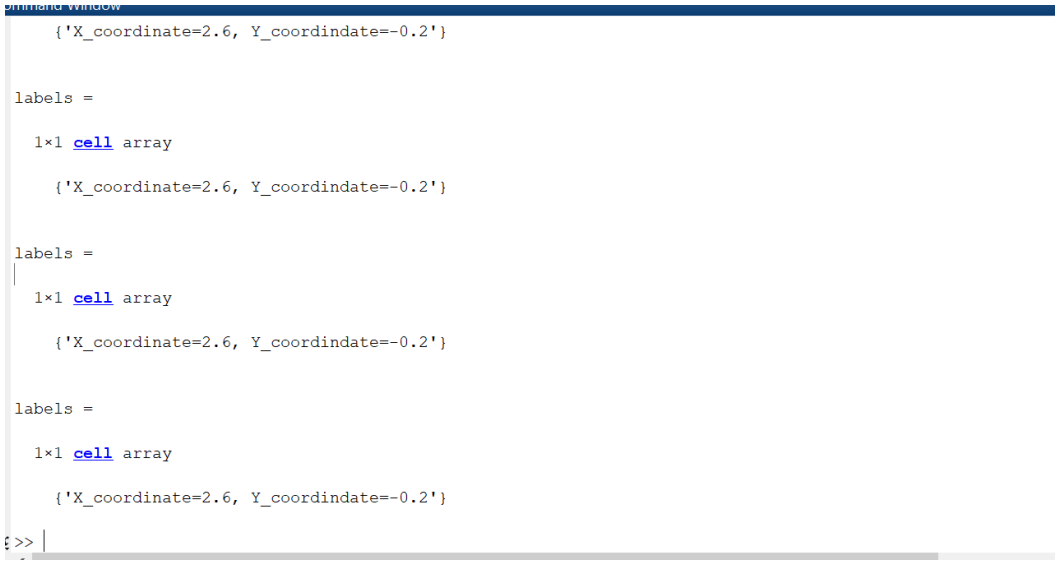

Figure 5.1: X and Y coordinates extraction

Using the list indexing the coordinates of the closest vehicle and the closest pedestrian is extracted. After the X and Y coordinate is extracted from the labels list, the distance between the coordinates of the ego vehicle and the objects detected is calculated by using the formula given in Equation 5.1

$$\text{Calculated Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad \text{- (5.1)}$$

According to the coordinate convention, the X axis is along the length of the car and the Y axis is perpendicular to the road direction. The coordinate of the ego vehicle is always set to (0,0). Thus, the equation 5.1 simplifies to equation 5.2

$$\text{Calculated Distance} = \sqrt{x^2 + y^2}. \quad \text{- (5.2)}$$

Finally, the distance between the ego vehicle and any obstacle is used in a “if” condition and according to the distance different measures will be taken. Fig 5.2 explains how the distance calculated will be used in the forward collision warning system

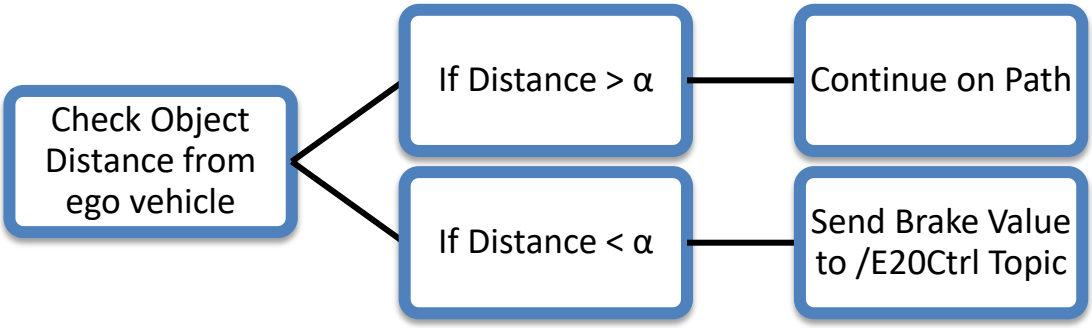

Figure 5.2 Forward Collision System Block Diagram

The above explained code is saved as car_ped_tracker.m code. Any reference to this MATLAB file, means a reference to section 5.1 which contains its working and it uses.

5.2 Pedestrian Tracking

Previously, as shown in Fig 5.3(a) and Fig. 5.3(b), only pedestrian detection had been focused upon. To make the full use of the MATLAB ADAS toolbox, tracking also was included. As explained in section 5.1, The car_ped_tracker.m code was used to detect the pedestrians in the environment, the working of the entire code is explained in section 5.1 and the full code is available in Appendix C.

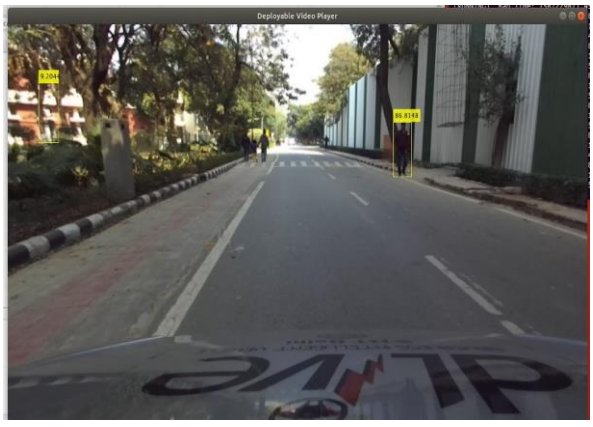

Fig 5.3(a)

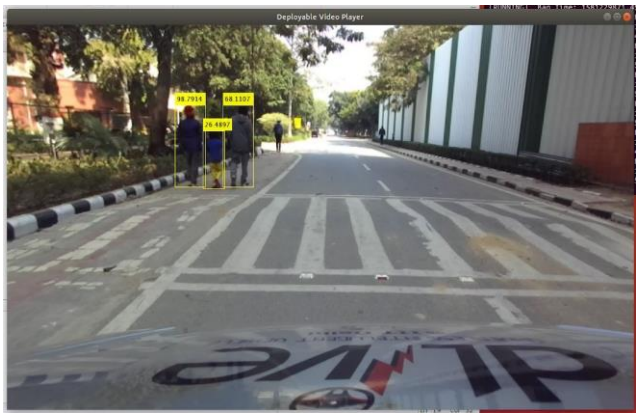

Fig 5.3(b)

Fig. 5.3(a)(b) Pedestrians Detection examples

Using the pedestrian tracker capabilities of MATLAB pedestrians can now be tracked and used for the forward collision system.

Fig. 5.4 Pedestrian tracking with coordinates wrt camera

Fig. 5.5 Multiple Pedestrian Tracking

In Fig 5.4 and Fig 5.5 , the coordinates of the pedestrians' wrt the car are obtained The ACF algorithm can detect several objects at once this making it very useful.

5.3 Vehicle Tracking

Like pedestrian tracking, an upgrade was made from vehicle detection to vehicle tracking. Fig 5.6(a) and Fig 5.6(b) are results of vehicle detection only. Using the car_ped_track.m code as explained in section 5.1, was able to detect moving or stationary vehicles in the surrounding environment.

Fig 5.6(a)

Fig 5.6(b)

Fig 5.6(a)(b) Vehicles Detection Example

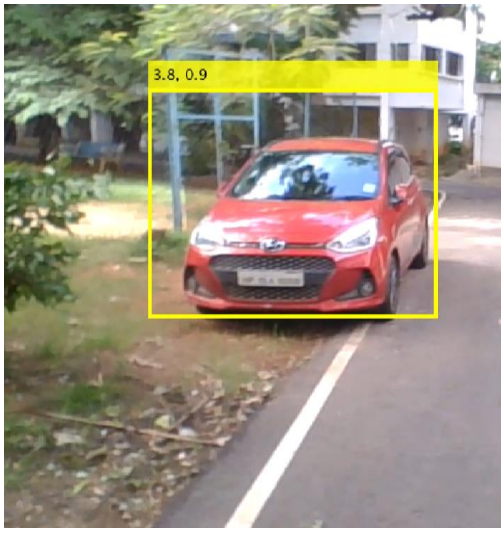

Fig 5.7 Vehicle tracking with coordinates wrt camera

Fig 5.8 Partially Hidden Car Tracking

In Fig 5.7 and Fig 5.8, the coordinates of the vehicle in world coordinates are obtained. Multiple vehicles can be detected using this code .

5.4 Collision Warning System

While lane driving just focuses on keeping the ego vehicle in between the lanes or maneuver in the drivable space. It is always possible that during lane driving any pedestrian or vehicle can come in the path of the car. Such situations are not common while driving, if suddenly the car ahead applies brakes or if a pedestrian runs across the road without looking for oncoming traffic. In such situations the Collison warning system will be useful. The monocular camera, which is the primary sensor, other than detecting lanes from the frames, will also be used to detect humans and vehicles for this purpose.

This system will thus detect the people or vehicles that come dangerously close to the ego vehicle. If this happens a ROS message will then be sent to the vehicle to apply brakes to maintain a safe distance. Because MATLAB and ROS[16] can have a bridge connection, this connection will be used to communicate between them. To send a message from MATLAB to ROS[16][17], the following bit of code must be used.

```
Sample_Publisher = rospublisher('/Topic_Name', 'std_msgs/Message_Type')
```

Sample_Publisher is a publisher that sends the required ROS[17] message type data to the Topic_name whichever is specified. The braking message can also be sent in different segments depending on the object distance. The ROS topic that receives the acceleration and braking In the E2o vehicle is the /E2octrl topic.

There are different cases to which the collision warning system will be tested. The videos for all these experiments were captured using the HP Wide Vision HD camera and the parameters are same as given in Table I

CASE I: To simulate a real-life scenario, a pedestrian was made to move in the same direction as the ego vehicle. The other vehicles in this example are stationary. In the forward collision system both the vehicles and pedestrians are detected simultaneously. The car_ped_tracker.m code is run which simultaneously detects and gives the distances of the detected objects.

Table ll shows the list with the coordinates of the detected vehicles and pedestrians. The minimum distance is calculated from the coordinates in the array is compared with the value of alpha. This calculation is explained in detail in section 5.1

Fig 5.9(a) Distance of pedestrians and vehicles more than alpha

Fig 5.9(b) Distance of pedestrians and vehicles more than alpha

Both the vehicles, as well as the pedestrians, are further than the alpha value set for this example. The alpha value set for the pedestrian, this is set at 5 meters, and for the vehicles, it is set at 3 meters. Note as the calculated distance for the closest pedestrian and the closest vehicle is more than the value of alpha set for both respectively, thus it is concluded that there is no imminent collision on course.

In this example as both the pedestrian and the vehicle are at a safe distance the annotation box argument, used in pedestrian_insert_box() and vehicle_insert_box() are set to be in green (Fig 4.9(a)(b)).

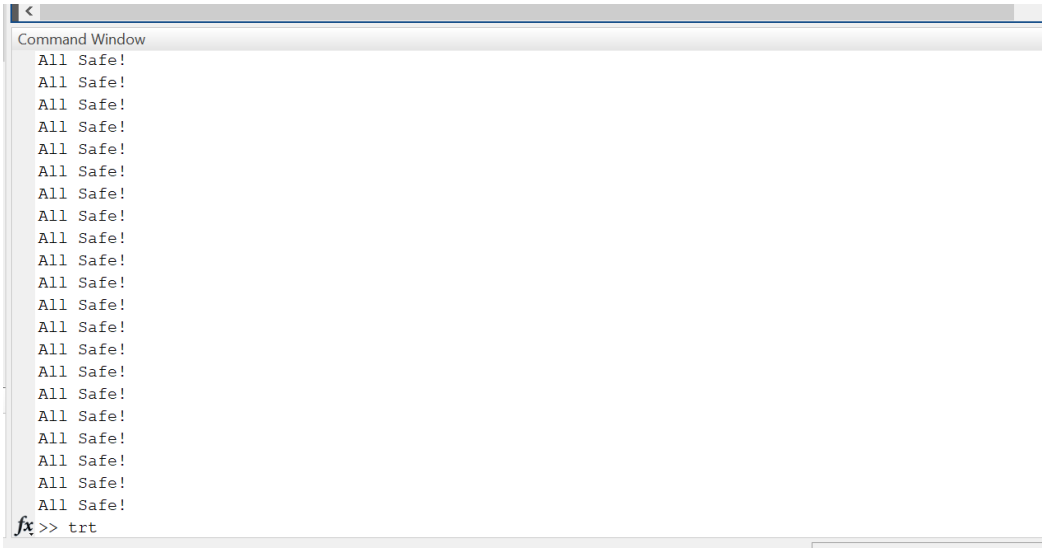

Fig 4.10 Status message Printed: No collision on track

TABLE II: Ego Vehicle Status for Case I

Figure Number	Pedestrian α Value	Vehicle α Value	Actual Pedestrian Distance	Actual Vehicle Distance	Ego Vehicle status
Fig 4.9 (a)	3 m	5 m	3.114 m	7.337 m	Safe
Fig 4.9 (b)	2 m	5 m	2.23 m	6.40	Safe

CASE II: In the same environment the ego vehicle now moves forward, and in doing so, the distance between the ego vehicle and the pedestrian decrease. The distance is again calculated by using the car_ped_tracker.m code. Table lll shows the list with the coordinates of the detected vehicles and pedestrians . The distance from the ego vehicle to the pedestrian now become less than the value of alpha set, aka the minimum distance after which a ROS message will be sent to the E2octrl topic to apply brakes and reduce the acceleration. The statement which will send the brake message to the ROS system will be executed in the pedestrian_insert_box() and vehicle_insert_box() function, as in these functions the exact distances are obtained where it could be used with an if statement to form different conditions.
Thus, there will be a pedestrian warning sent via ROS messages to the central control system to slow down or stop to avoid a collision with the pedestrian. Note as the calculated distance for the closest pedestrian is less than the value of alpha set for the pedestrian, thus it is concluded that there is imminent collision on course and that too with a pedestrian.

Fig 5.11 (a), The pedestrian, is closer to the ego vehicle than the threshold safe distance (3 meters)

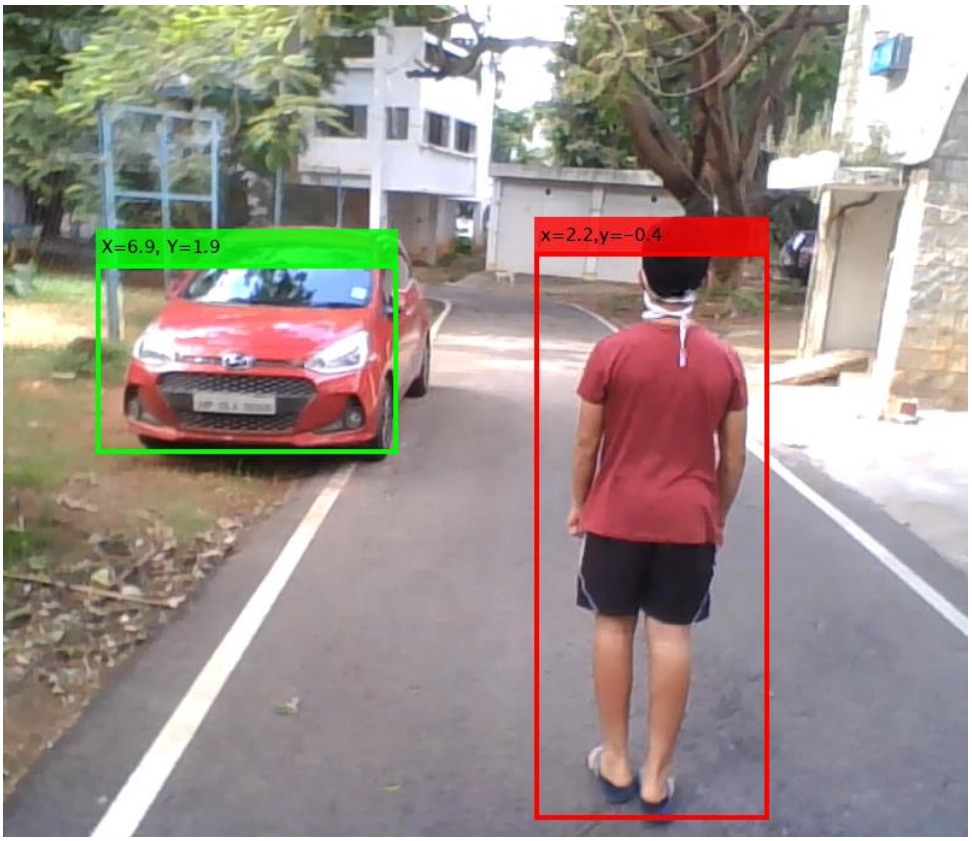

Fig 5.11 (b) The pedestrian, is closer to the ego vehicle than the threshold safe distance (3 meters)

In this example as the pedestrian is at a distance lower than the alpha value set, in annotation box argument, used in pedestrian_insert_box() is set to color red to denote a warning and vehicle_insert_box() is set to be in green as the vehicle is still quite far off (Fig 5.11(a)(b)).

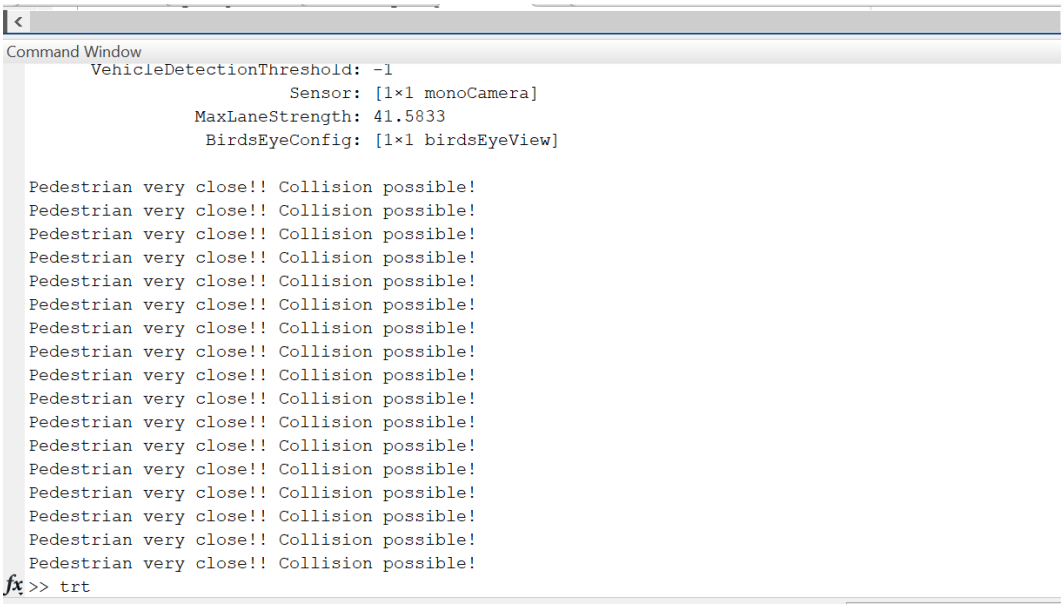

Fig 5.12 Status message Printed: Pedestrian Very close, Possible collision

TABLE III: Ego Vehicle Status for Case II

Figure No.	Pedestrian α Value	Vehicle α Value	Actual Pedestrian Distance	Actual Vehicle Distance	Ego Vehicle status
Fig 5.11 (a)	3 m	5 m	2.86 m	7.337 m	Collision with Pedestrian
Fig 5.11 (b)	3 m	5 m	2.23 m	6.40	Collision with Pedestrain

CASE III: In this case the scenario is that instead of a pedestrian a vehicle might move closer to the Ego vehicle, and the distance between the ego vehicle and the detected vehicle might become less than the value of alpha set, which is the safe threshold distance set (Fig 5.13(a)(b)). Table IV shows the list with the coordinates of the detected vehicles and pedestrians. The distance is again calculated by using the car_ped_tracker.m code. If such a scenario arises a ROS message will be sent to the E2octrl topic to apply brakes and reduce the acceleration. The statement which will send the brake message to the ROS system will be executed in the pedestrian_insert_box() and vehicle_insert_box() function, as in these functions the exact distances are obtained where it could be used with an if statement to form different conditions. Thus, there will be a vehicle warning sent via ROS messages to the central control system to slow down or stop to avoid a Collison with the vehicle.
Note as the calculated distance for the closest vehicle is less than the value of alpha set for the vehicle, thus it is concluded that there is imminent collision on course and that too with a vehicle.

Fig 5.13 (a) The vehicle is closer to the ego vehicle than the threshold safe distance (5 meters)

Fig 5.13 (b) The vehicle is closer to the ego vehicle than the threshold safe distance (4 meters)

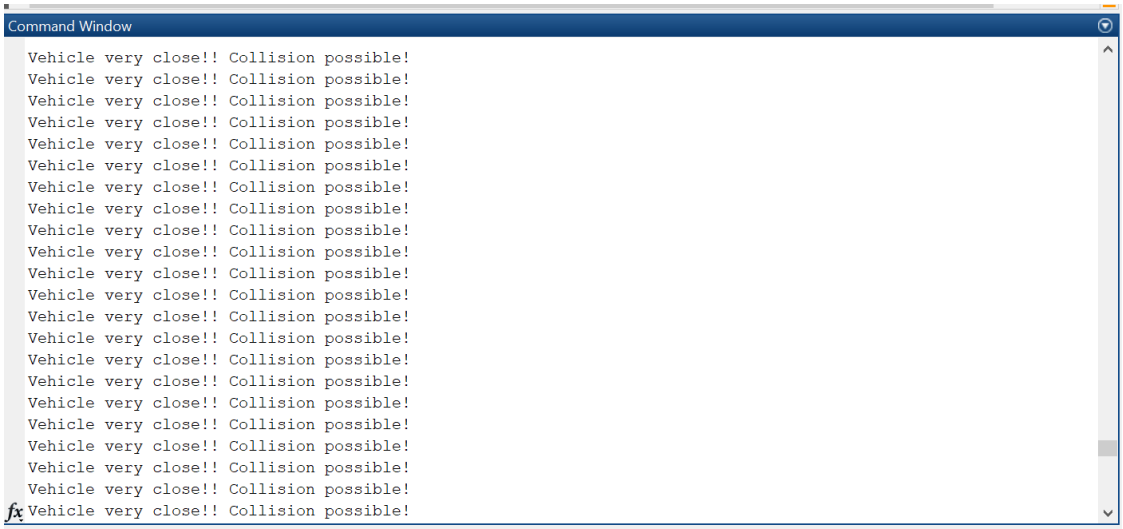

Fig 5.14 Status message Printed: Vehicle Very close, Possible collision

TABLE IV: Ego Vehicle Status for Case III

Figure No.	Pedestrian α Value	Vehicle α Value	Actual Pedestrian Distance	Actual Vehicle Distance	Ego Vehicle status
Fig 5.13(a)	2 m	5 m	2.94 m	2.64 m	Collision with Vehicle
Fig 5.13(b)	-	4 m	-	3.64 m	Collision with Vehicle

Chapter 6: Path Planning Simulations

6.1 Simulation in MATLAB

The automated parking valet system[18] was explored in which various techniques that could assist path planning were worked upon. The full code for this simulation can be seen in the appendix D of this thesis. For this example, the default map of a parking area which was with occupied parking spots, road markings was used as shown in Fig 6.1. This map is an occupancy map that is built using Simultaneous Localization and Mapping. For this example, the map is already provided.

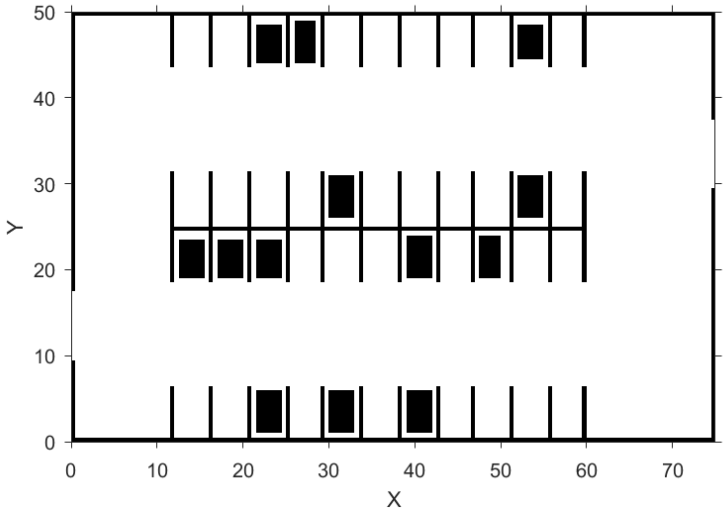

Fig 6.1 The Parking Lot Occupancy Map

The Vehicle_Dimension object was used to make the dimensions of the simulated vehicle in accordance with the actual car. The different parameters were changed according to the E2o vehicle. The following table denotes the values that were used for simulation

TABLE V: Dimensions of the vehicle

Length	3280 mm
Width	1514 mm
Height	1560 mm
Wheelbase	1958 mm
Turning Radius	3900 mm
Steering Angle	35 Degs

Fig 6.2 shows all dimensions that be controlled via this object.

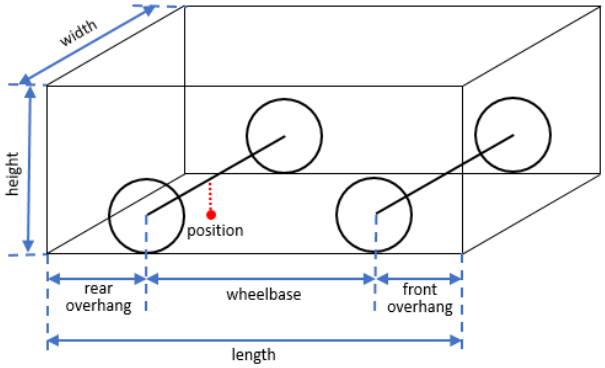

Fig 6.2 Simulated Vehicle Dimensions

As shown in TABLE V, the width, height, length, wheelbase of the Mahindra E20 vehicle was set according to the actual values.

The vehicle_current_Pose() object was then used to define the starting orientation of the vehicle in the map. The initial pose of the vehicle was set to [4 12 0]

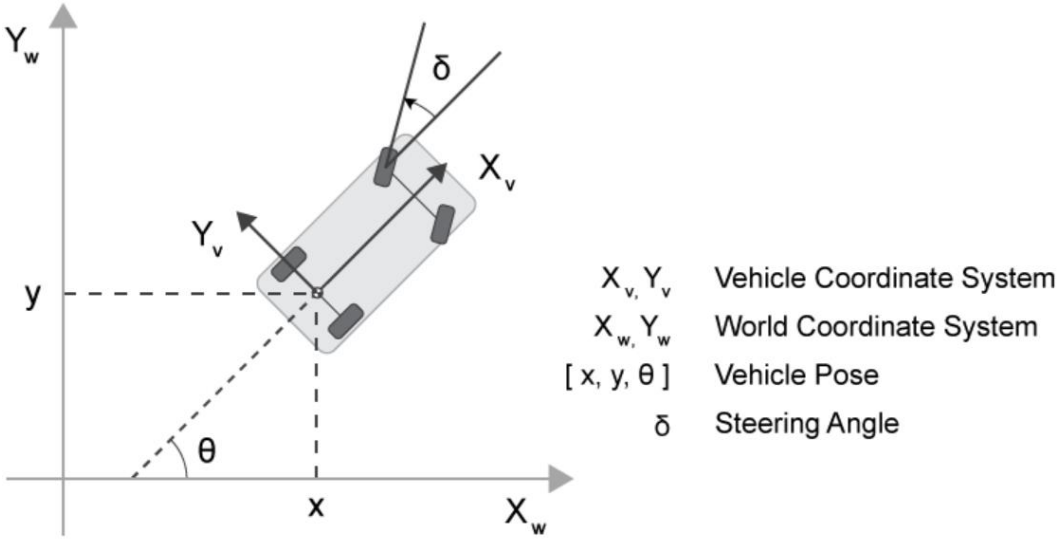

Fig 6.3 Relation between vehicle and world coordinate frame

The pose of the vehicle is defined in the world coordinates[19]. The full pose of the vehicle is given by [x,y, theta]. Where x,y give the place where the center of the car's rear axle is located with respect to the world coordinate system. The orientation/heading of the car is given by the value theta, which is also with respect to the X axis of the world frame (Fig 6.3).

The whole path is divided into many goal points which the vehicle had to transverse. For this example, a static goal route plan was used.

TABLE VI Start and End pose of the vehicle in the map

Start Pose			End Pose		
X	Y	Theta	X	Y	Theta
4	12	0	56	11	0
56	11	0	70	19	90
70	19	90	70	32	90
70	32	90	53	38	180

The vehicle was then plotted at the locations as shown in Fig 6.4 which were mentioned in Table VI. The start and end pose, coordinates and theta are in the world frame, which in this case is the map frame.

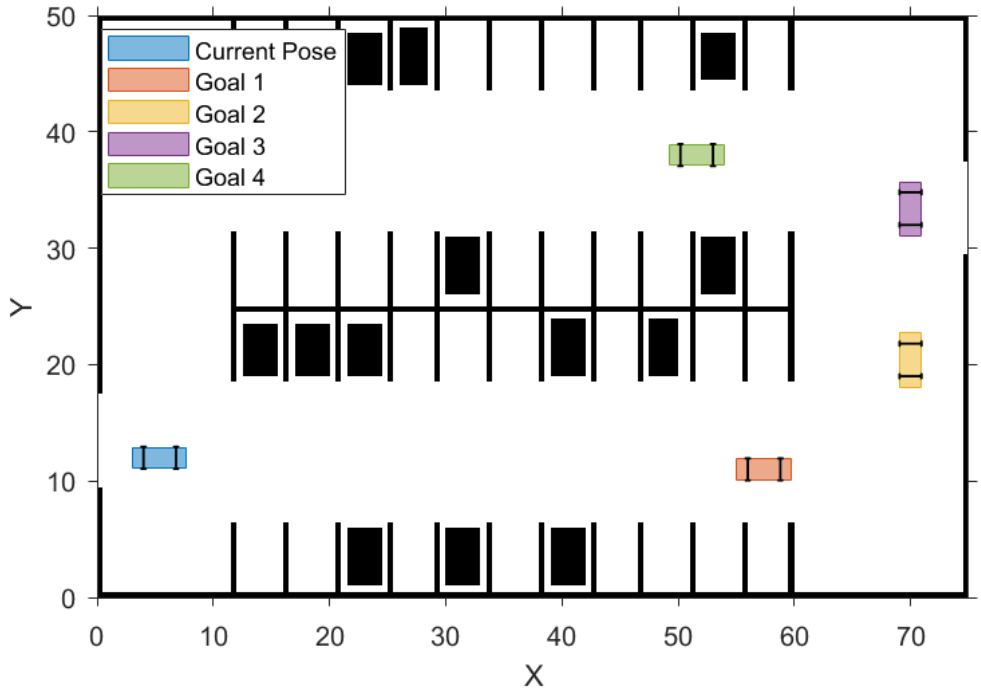

Fig 6.4 Vehicle plotted at goal locations

The Main idea as to how the car will move in between the goal points is

a. **Motion Planning** - The Rapidly Exploring Random Tree(RRT)[20][21] technique is used to find a suitable path for the vehicle. For this the path_planner_RRT() function is invoked. Using this function, a path planner object motion_vehicle_planner is made to create a path between a starting point and a goal point. The RRT path planner explores the occupancy map that is provided. The occupancy map gives a sense of which areas are free and which are occupied. The RRT path planner by using this map builds a series of collision free paths in the direction of the goal point. The main input parameters to the path_planner_RRT are

TABLE VII Parameters used for RRT planner

Parameter	Value
Cost Map	Occupancy map provided
Min_Iterations	1000
Connection_Distance	10 m
Minimum Turning radius	3.9 m

The minimum_Iterations parameter sets the minimum number of times the planner will go about the cost map and explore it to find feasible paths in the map.
The connection distance parameter refers to the distance between two feasible points that are given by the path planner. The larger this value the longer will the path between 2 node points.

Minimum turning radius is achieved when the steering is set at the maximum, then the radius of the circle that the car takes while this configuration is defined as the Minimum turning radius.

b. **Path Smoothing** – The path created by the RRT path planner might not always be smooth, due to this there might be sudden changes in the steering to follow such a path. Such paths will be highly inconvenient for the user and the path between any two points should be as such that the path is always differentiable. A cubic polynomial spline[22][23] is tried to fit between the points to make the path smoother (Fig 6.5). The smooth_path_spline() function is used for this. This function gives the smoothed poses which are on a differentiable path.

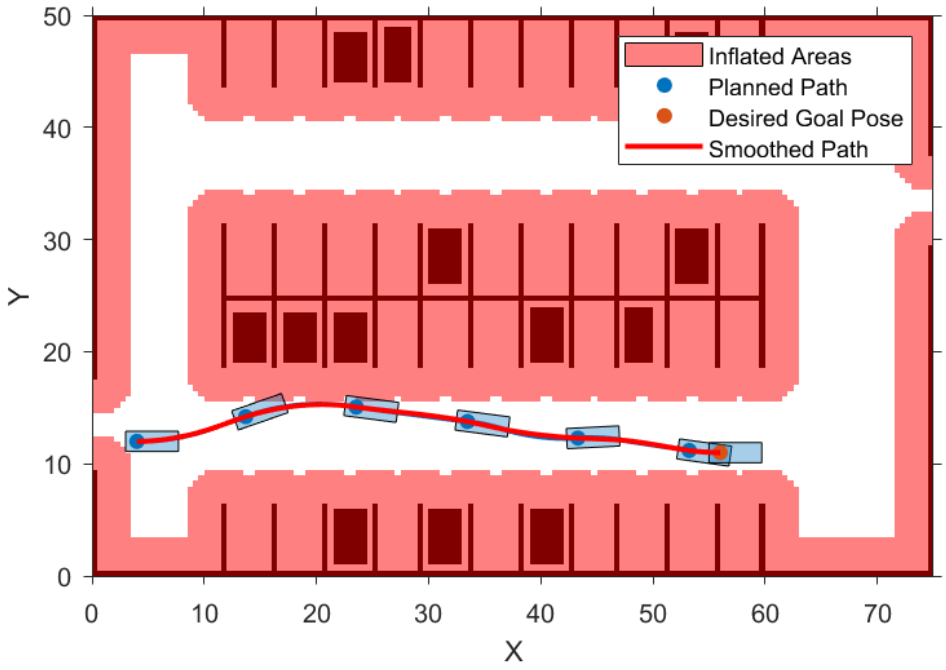

Fig 6.5 Vehicle moving on a smoothened path

c. **Vehicle Control**- Given a smoothed reference path, the reference pose and velocity is calculated on the current pose and velocity of the vehicle. The steering angle is calculated. For this the bicycle kinematic model[24][25] is used.

Lateral Control – Steering control of the vehicle

Bicycle kinematic model

In this model first a reference point is selected. In this example the reference point is taken as the center of the rear axle. The center of the rear and front axle is replaced by single wheels. The location of the rear axle is taken as (xr,yr) and the direction of heading of the bicycle is denoted by theta. The length of the vehicle in this case bicycle is denoted by L. A steering angle, delta, for the front wheel is defined (Fig 6.6), it is measured relative to the direction in which the bicycle is moving.

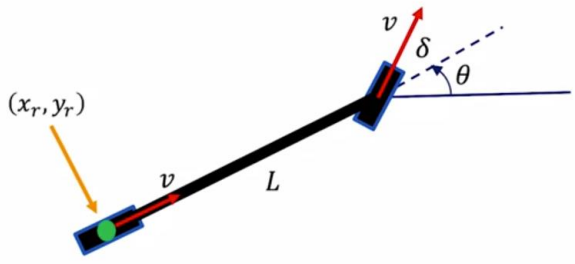

Fig 6.6 Rear Axle Bicycle Model

The speed for each tyre is denoted by v and is in the direction of the wheel, this condition is the no slip condition Because of this condition, the rate of rotation of the of the bicycle which is omega, is given by velocity / Instantaneous center of rotation, with a radius of R. By using simple trigonometry, the relation between the steering angle delta and the rate of rotation of the bicycle is calculated.

L: The Length of the vehicle(bicycle) between the midpoint of the front and rear axle

R: Instantaneous turn radius

V : velocity in the direction of the tyres

Delta : Steering angle

Theta: heading of the Bicycle

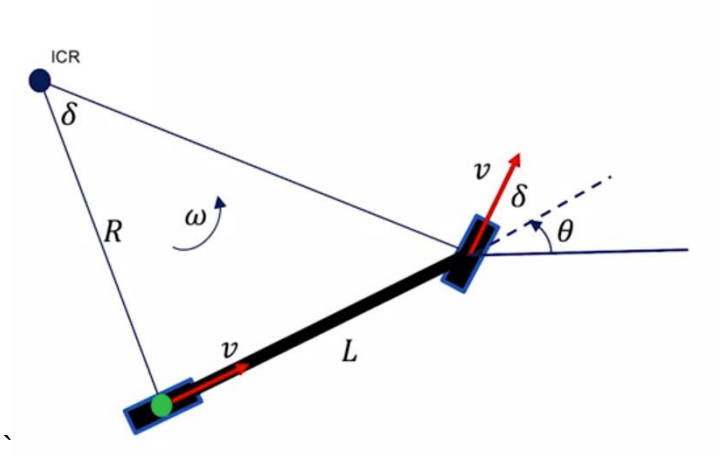

Fig 6.7 Model Analysis of Rear Axle

$$\dot{\theta} = \omega = \frac{v}{R} \quad \text{- (6.1)}$$

As the no slip condition is valid omega can be directly be equated with velocity/ R.

$$\tan\delta = \frac{L}{R} \quad \text{- (6.2)}$$

Using simple trignometry relations it can be seen in figure 6.7 that tan of delta is L/R

By using both equation 6.1 and 6.2 the following equation is conceived

$$\dot{\theta} = \omega = \frac{v}{R} = \frac{v \tan \delta}{L}$$ - (6.3)

After this the whole kinematic model can be computed. The velocity components of the referance points are then given by

$$\dot{x}_r = v \cos \theta$$
$$\dot{y}_r = v \sin \theta$$
$$\dot{\theta} = \frac{v \tan \delta}{L}$$

` - (6.4, 6.5, 6.6)

Where xr(dot) and yr(dot) give the velocity of the reference point, in relation with the velocity and the heading theta. Thus, a direct relation between the steering angle delta and the rotation rate of the vehicle omega is obtained.

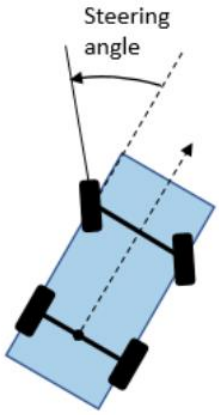

Fig 6.8 Steering angle relative to vehicle heading

The lateral control is given by the lateralcontrollerstanley() function. This function makes use of the above-mentioned bicycle model an calculates the steering angles required to stay on the path as given by the planner. Thus, the function to calculate the steering, the controller tries to minimize the position error, and the associated angle error of the current orientation wrt to the reference orientation.

Longitudinal Control

This Longitudinal control[20] then calculates the acceleration and deceleration values, to calculate the desired velocity of the vehicle at a point. For this example, the reference velocity must be known, along with the current velocity and the heading. Using these parameters, the controller tries to minimize errors. The acceleration and deceleration of the vehicle is then calculated using the Longitudinal Controller function. It implements a Proportional Integral controller. The equation which is used for this is

$$u(k) = (K_{\mathrm{p}} + K_{\mathrm{i}} \frac{T_{\mathrm{s}}\, z}{z-1})\, e(k)$$

- (6.7)

TABLE VIII: Parameters in the PI controller

Variable	Definition
K(p)	Proportional Gain value
K(i)	Integral Gain Value
T(s)	Sample time of the control block set in seconds
e(k)	The error in velocity, that is the difference between the current velocity v(k) and the desired velocity at that point d(k) = v(k) – d(k)
u(k)	The control step at the k time step

The calculated control step u(k), is used to calculate the acceleration and deceleration of the vehicle. The algorithm puts a limit to the maximum value of both acceleration from [0,Ac] and deceleration from [0,Dc].

Ac is the maximum acceleration in the longitudinal direction(in m/s^2)

Dc is the maximum deceleration in the longitudinal direction(in m/s^2)

During an iteration only one of the above can be non-zero, the other is therefore set to 0. This prevents the vehicle from accelerating and decelerating simultaneously

d. **Goal Checking**- Till the vehicle has not reached its goal position the above 3 steps are repeated in a loop. As soon as the vehicle reaches its goal (Fig 6.10(a)(b)) point the loop is terminated .

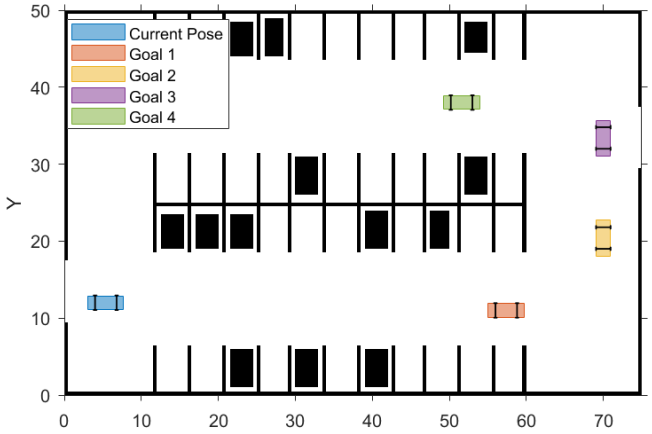

Fig 6.9(a) Vehicle at various goal points

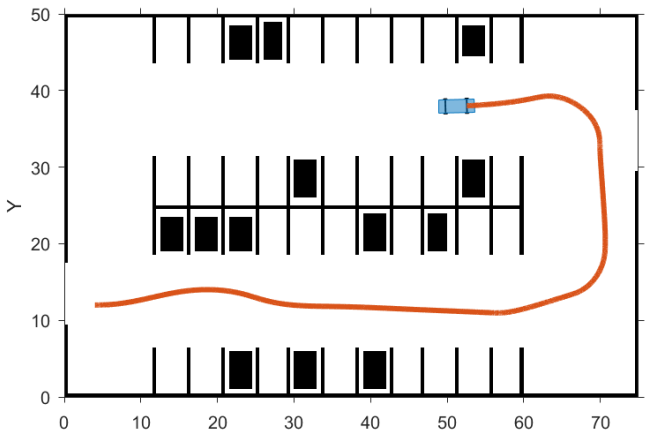

Fig 6.9(b)Vehicle after traversing all points

Chapter 7: Miscellaneous Functions

Other domains were also inspected, and analysis was done on the the implementation they could have in this project. In this chapter, other topics which include vision shall be focused upon. One is using CNNs for traffic sign detection, and the other one is to create an occupancy grid.

7.1 CNN Based Traffic sign Detection

A Keras based CNN[25][26][27] model was built to classify traffic signs from the live camera feed. The underlying motivation comes from the fact that the autonomous vehicle should able to detect and must abide by the traffic rules. While driving on the road, if the vehicle faces some traffic signs which can potentially harm either the vehicle or can cause damage to human or public property, the vehicle shall act accordingly.

The problem at hand is an image classification problem. The road signs which are encountered while driving must be classified into different categories so that when the vehicle detects any one of them, it can act accordingly. For the classification, 43 different kinds of road signs were used. These road signs were the common ones found almost everywhere. The total dataset comprised of more than 35000 images, belonging to these 43 classes. More than 600 images per class were used for training and the individual size of each image was 32 pixels by 32 pixels, with 3 color channels
The road sign classes are as follows:-

TABLE IX Road Sign Classes

Class ID	Name of the Road Sign
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for Vehicles over 3.5 ton
11	Right-of-way at next intersection
12	Priority Road

13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 ton prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road Narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	Ending of no passing for vehicles over 3.5 tons

Data preprocessing – After importing the dataset, It was then to be split in training set, test set and Validation set. 20% of the total images were split into the test set and 20% of the remaining were split into the validation set. The training data has the following shape (22271, 32, 32, 3), which corresponds too 22271 images each of shape 32X32X3 (32 pixels by 32 pixels by 3 color channels). Similarly, the validation set had the shape

(5568, 32, 32, 3) and the test set had the shape (6960, 32, 32, 3).
All the dataset was then processed before feeding it to the neural network. The images were converted to grayscale using OpenCV, then the lighting was standardized in the image by using the equalize histogram function of OpenCV. Finally, each pixel value in the 32 X 32 sized image was divided by 255 or was normalized so that each value would be between 0 and 1. After the above preprocessing the shape of each of the set changed. This is because as the image is converted into gray scale it ceases to have 3 channels. The Training set thus became (22271, 32, 32) in shape. Again, the shape of each set was changed and a depth of 1 was added to it, so that it could be fed to the network.

CNN Model - Any existing model like VGG16, VGG19, and ResNet was not used because it took very long to train. Thus a 10-layer neural network was used for this task. The model layers is defined as follows

TABLE X Model Architecture

Layer (Type)	Output Shape	No. of parameters
conv2d_1 (Conv2D)	(None, 28, 28, 60)	1560
conv2d_2 (Conv2D)	(None, 24, 24, 60)	90060
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 60)	0
conv2d_3 (Conv2D)	(None, 10, 10, 30)	16230
conv2d_4 (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d_2 (MaxPooling2D	(None, 4, 4, 30)	0
dropout_1 (Dropout)	(None, 4, 4, 30)	0
flatten_1 (Flatten)	(None, 480)	0
dense_1 (Dense)	(None, 500)	240500
dropout_2 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 43)	21543

Total parameters: 378,023
Trainable parameters: 378,023
Non-trainable params: 0

The model Learning rate was set to 0.001. and for calculating the loss the categorical cross entropy loss, was used. As there was to be only a single label against each image, that is why this loss cross entropy was used.
The Cross-Entropy loss is given by the following formula -:

$$-\sum_{i}^{C} \text{gi} * \log(\text{mi}) \qquad \text{- (7.1)}$$

Where g_i and m_i are the ground labels or the ground truth and the CNN model output for each individual class i in C.

For the categorical cross entropy loss, a SoftMax function was applied to the scores of the model before feeding it to the Cross-entropy loss function.

$$f(m)_i = \frac{e^{m_i}}{\sum_{j}^{C} e^{m_j}} \qquad \text{- (7.2)}$$

Where m_i and m_j are the scores or outputs given by the neural network for each class in C. The SoftMax function changes all the scores as probability which adds up to one for all the classes.
The model was run for 10 epochs, with steps per epoch set at 2000. The batch size set was 50. The training was conducted on a home workstation with Intel i7 8550U processor, 8 GB RAM with intel internal 610UHD Graphic card. As the image size was only 32X32, the training was done on CPU rather than GPU. The training took close to ~90 mins. The loss and accuracy vs epochs graphs for training and validation sets are given in Fig 7.1 and Fig 7.2.

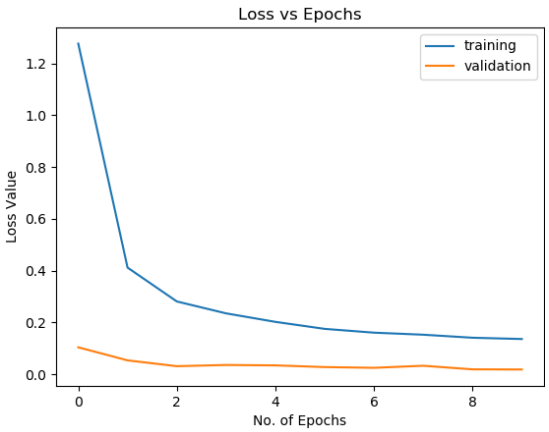

Fig 7.1 Loss value vs. epochs

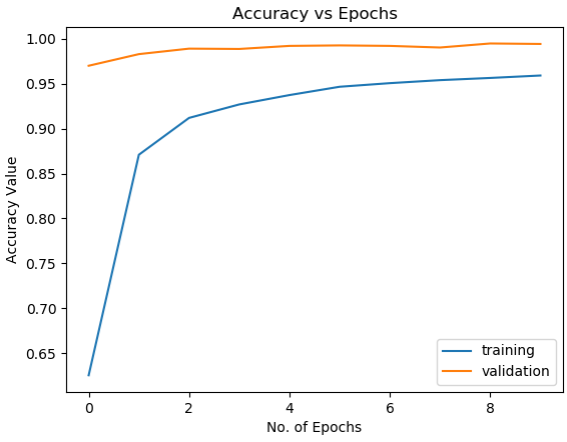

Fig 7.2 Accuracy Value vs. epochs

The model was run against the test set to get the classification report of the model. The confusion matrix was calculated for the 43 classes and using it the accuracy metrics

were evaluated. There are different evaluation methods that can be deployed for the model, the ones that are used are

1. Precision- This is the proportion of rightly predicted positive subjects to the total predicted positive subjects. The formula given is (Number of True Positives)/(Number of(True Positives +False Positives))
2. Accuracy – It is simply the ratio of rightly predicted subjects to the total subjects. The formula is
   Accuracy = Number of (True Positives +True Negatives)/(Number of (True Positives + False Positives +False Negatives +True Negatives))
3. Recall- It is the proportion of rightly predicted positive subjects to the all subjects in actual class. The formula is (Number of True Positive) /Number of (True Positive + False Negative)
4. F1 Score - the F1 Score is the weighted mean or average of precision and recall. The formula is 2*(Recall value * Precision value) / (Recall value + Precision value)

The results are tabulated in a table below.

TABLE XI Model Results

Accuracy	Precision	F-1 Score	Recall
0.995	0.99	0.99	0.99

The model is judged based on its F1 score this is because. The F1 score takes both the false negatives and false positive into account. F1 score is much more complicated than simply finding out the accuracy. But this score gives more comprehensive results than by just using the precision or accuracy.
Live video tests were also performed by using the inbuilt webcam of the laptop. The webcam model is not relevant to the model, any webcam can be used for testing. The model was able to detect and classify the images that were shown. The accuracy of the model was tested by running the test set on the model, and the Test Accuracy came about ~0.995.

In the experiment the validation accuracy is higher than the training accuracy. This is because the dataset which is used for this experiment might not have the split which should be done in random. There still might be some sorts of patterns in the validation set which the computer is able to learn. Thus, if a pattern emerges the dataset tends to have less noise and for the model it becomes very easier to detect an image.
Using the output of the CNN model, that can specify an action according to the detected road sign. For example, if a stop sign is detected, a ROS message can be published to

the velocity control to stop and thus apply brakes as shown in Fig 7.3.

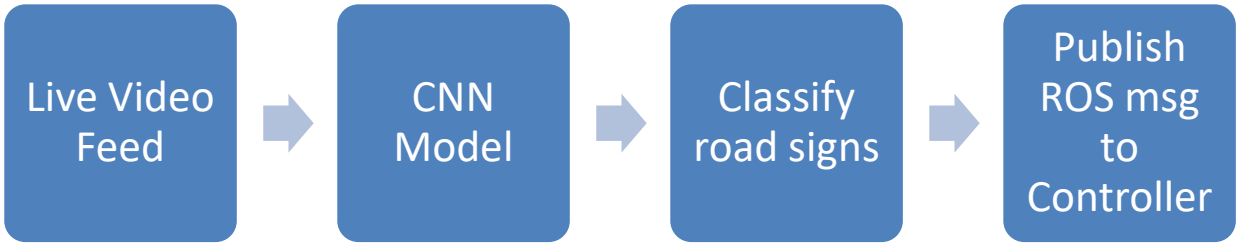

Fig 7.3 Workflow for the implementation of CNN model

(a)

(b)

(c)

(d)

Fig 7.4(a)(b)(c)(d) Different road sign detection examples

7.2 Creating an Occupancy grid using Monocular Camera

The vehicle cost map is one of the most important aspect to investigate the feasibility of the path made. If the points generated by the path planner collides or intersects with any obstacle on the map, the path is then dropped. For this example, a pretrained semantic segmentation network, a deep CNN architecture (SegNet with a VGG-16 encoder)[28][29] is used. The network returns classifications for each image pixel in the image. That is each pixel in the image will be run in the neural network and each pixel will be allocated a class, in this example only those pixels which represent a road are used. The free space is identified as image pixels that have been classified as Road.
All the images in this section were taken using the HP Wide Vision HD camera with the same parameters as given in Table I.

A single frame from a video or an image is taken and is feeded to the semanticseg() function, which segments the image into the 11 classes on which it is trained. The semanticseg() is a deep neural network which is pretrained. Each pixel in the image will be run in the neural network and each pixel will be allocated a class. The network runs on the image and gives the output based on the training of the model. Then the free space is shown as the annotated image (Fig 7.5).

Fig 7.5 Free drivable space

At this point the road segmentation is just visual, it has no use for any path planner which might its information to check the feasibility of the path in the frame. Thus, the confidence in the free space must be calculated. This value will be used to inform the algorithms about the neural network's estimate validity. The freeSpaceConfidence() function converts the pixels in the frame into confidence values. The function assigns confidence value to each pixel. The surer the network is of free space the higher confidence it assigns the pixel. The highest value that the function assigns is 1, meaning it is 100% confident that the pixel is of free space, if the value of 0 gets assigned to the pixel it means that the model is 100% sure that the pixel is not of free

space aka that part does not comprise of the road. Thus, each pixel which was earlier used to demonstrate the free space goes through this function only to be assigned a confidence score (Fig 7.6). This may be useful, for example, even if the network classifies a pixel as Road, the confidence score may be low enough to disregard that classification for safety reasons.

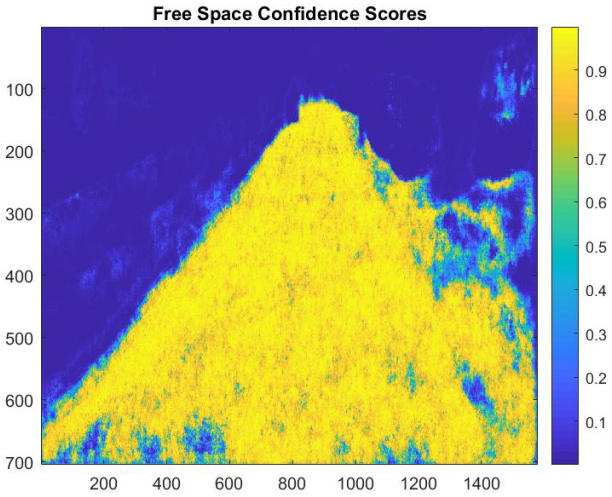

Fig 7.6 Free Space Confidence score

The output of the confidence function gives us the surety that the road is either drivable or has several obstacles. The confidence ranges from [0,1], where 1 corresponds to drivable space, and 0 means obstacle. This map gives us confidence in the pixels of the image rather than the vehicle coordinate system.

The confidence image which was created earlier was made in the image space which is not useful for the navigation system. The image thus should be converted into world space. Occupancy grids are usually implemented to show a vehicle's surroundings as a grid in vehicle coordinates and are used for path planning This is typically done by transforming the image into bird’s eye view image.
Using the same camera and parameters as before. The birdsEyeView() function is used to transform the image into BEV (Fig 7.7). The freeSpaceConfidence() function is also applied then to the BEV image to get the free space confidence in the world coordinate system. That is each pixel in the BEV image is assigned a value between [0,1], where 1 means 100% no obstacle and 0 means 100% obstacle (Fig 7.8).

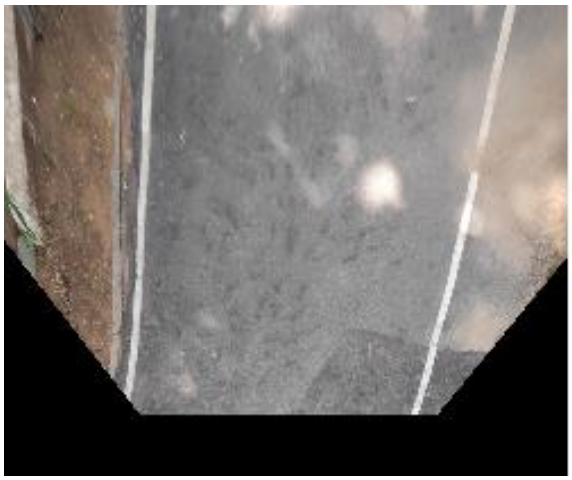

Fig 7.7 Birds Eye View of the Road

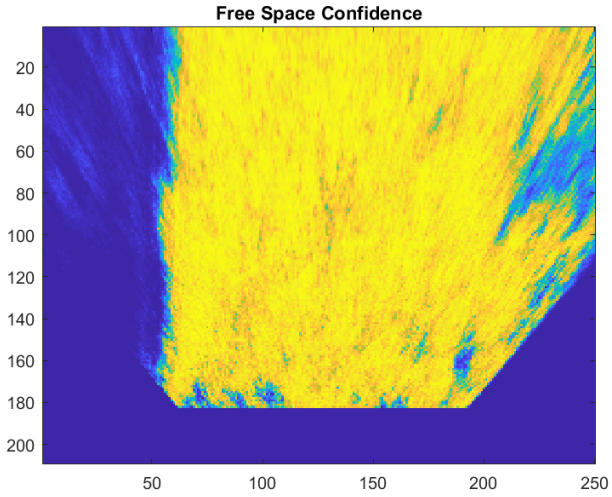

Fig 7.8 Free Space Confidence in BEV

After the BEV confidence image is created the createOccupancyGridfromFreeSpace() function creates an occupancy grid based on free space estimation (Fig 7.9). The difference between the free space confidence image and the occupancy grid is that, the former is in image frame of reference that is it is shown or represented in pixels, but the occupancy grid is represented in distance in meters in the world coordinate frame. The coverageAreaPlotter() might be used to show the sensors on the BEV image .

The map is oriented in such a manner that it lines up with vehicle coordinates, where X is along the length of the vehicle, and the Y-axis is perpendicular to the direction of the road

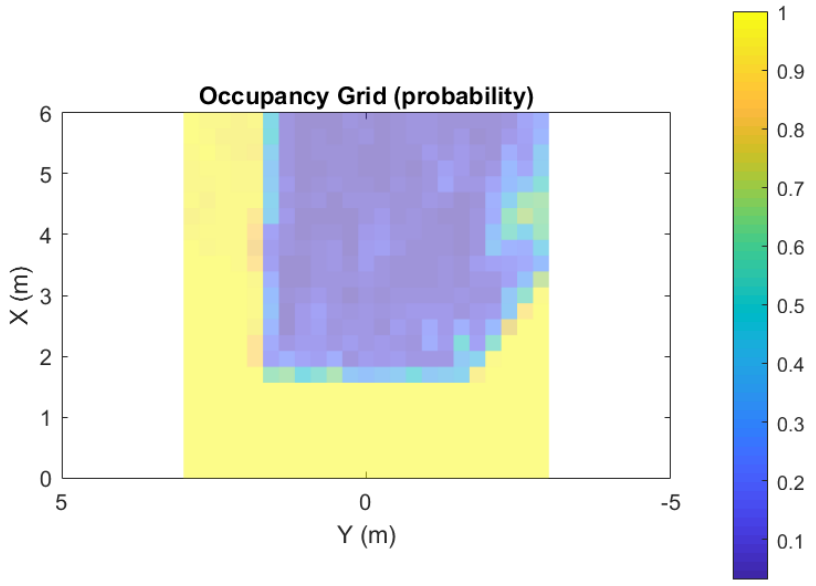

Fig 7.9 Probability of free space in the vehicle coordinate

The vehicleCostMap() function is used to convert the occupancy grid into a vehicle cost map. This function brings the world coordinates into the map as well. Using the output image generated different locations in the world system can be inspected without going to that point. The confidence can easily be used to determine whether the point on the map is free or has obstacles. A threshold of probability can be set above which the point can be safely assumed to be free.

To check the usefulness of the map, a set of points are created in the in the world coordinate. These locations can represent the path the car might travel. checkOccupied() function is used to determine the feasibility of the points, whether each location is occupied or free. The function plots each point on the vehicle cost map and checks the confidence value of that point, if the confidence is greater than a safe threshold then the function gives it a safe allocation else it is given a false one. For this example, a confidence of more than 0.7 was used. Based on the results, a potential path might be impossible to follow because it might go out of bounds of the road and might collide with obstacles

TABLE XII Status of Test points on the map

Point No.	X Coordinate	Y coordinate	Status(Free/Occupied)
1	5	0.375	Free Space
2	4	0.375	Free Space
3	4	2	Occupied Cell
4	3	0.375	Free Space
5	2	1.7	Occupied cell

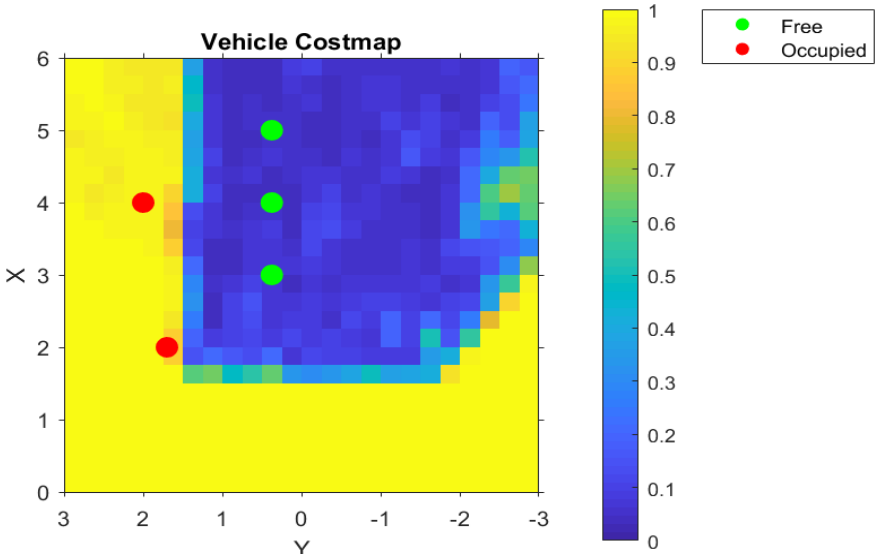

Fig 7.10 Acceptable Path points on the road

The Coordinates from the TABLE XII are marked in Fig 7.10. The point number 1, 2 and 4 are marked safe with no obstacle in the path whereas the points 3 and 5 are marked unsafe as they are colliding with area outside of the road.

In Fig 7.10, The points which are on the drivable space can be seen, which is the road, are marked with green whereas the points where the road ends and is not present are marked in red.

7.3 CARLA WAYPOINT FOLLOWING

Carla is a simulator to develop algorithms for vehicles in the field of autonomous driving. CARLA was developed by people from the CV center at the Autonomous University of Barcelona, and a team from Intel and Toyota Research Institute. CARLA was built using the Unreal game engine.

CARLA can replicate real life scenarios in a simulation environment. Simulating the vehicle and checking all the algorithms are very important and necessary before the vehicle is used for driving. Thus CARLA helps as a tool for testing the different algorithms on different cases on the road. Almost all primary sensors which are required for autonomous driving can be simulated and sensor data can be used for development. Lidar, Camera based systems can thus be tested on the CARLA platform.

In Fig 7.11 a CARLA environment has been launched, the environment comprises of roads with details such as lane markers, zebra crossing and dashed lines.

Fig 7.11 CARLA default environment

The environment also has working traffic lights, and can add more vehicles and pedestrians to make the environment very complex (Fig 7.12).

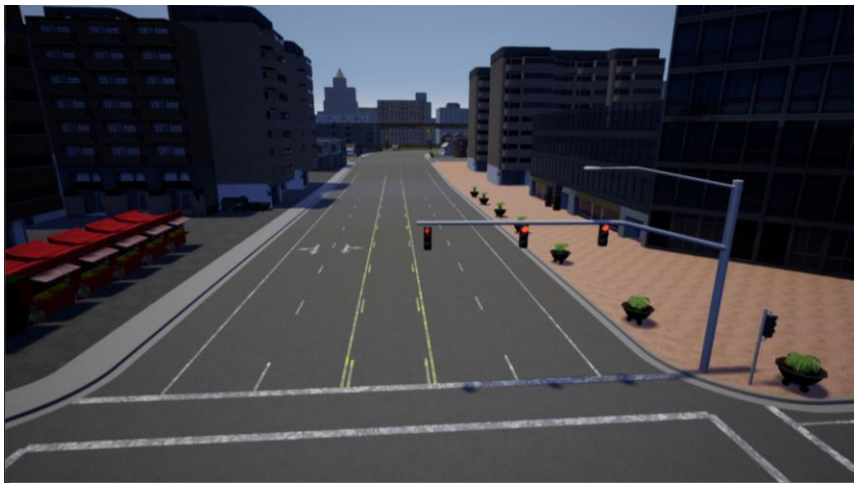

Fig 7.12 CARLA default environment with no vehicles

In CARLA the entire simulation is controlled using an external client which serves as connection with the CARLA server. The external client can thus send or receive messages which can be used to control the car or take in sensor data.The main demonstration is done on CARLA server mode, in this mode a python script acts as the external client and sends messages to the vehicle and controls its steering and throttle

Fig 7.13 Vehicles simulated in the CARLA environment

The vehicle will be launched in a predefined map and lateral and longitudinal control shall be implemented on the vehicle. As there are no vision sensors or lidar sensors used, the track is unpopulated with pedestrians and vehicles. The vehicle shall move from the initial point to the end point of the track. While moving there is a list of predefined way points given which contain the x, y coordinates of the way point on the track and the desired velocity that the vehicle should have at that point. These act as reference signals for the controllers which are to be used. A lateral controller is implemented for steering and Longitudinal is implemented for throttle and brake control.

Implementing Controllers in CARLA

Longitudinal Controller

Just like in MATLAB the longitudinal controller uses a PID controller to calculate the Acceleration and brake of the vehicle running on the track. The PID controller used consists of three components. The PID controller applied in the code is given in equation -

$$u = K_P(v_d - v) + K_I \int_0^t (v_d - v)dt + K_D \frac{d(v_d - v)}{dt} \quad \text{- (7.3)}$$

where u is the acceleration input to the vehicle
Vd is the desired speed
V is the actual speed

Three types of errors and Gains are used to build this PID controller

1. Speed Error – In this error a pure gain Kp is used that scales the vehicle acceleration based on the speed error, which is the difference between the desired and actual velocities.
   This makes sure that the ego vehicle accelerates in the correct direction with a magnitude proportional to the error.

$$\boxed{K_P(v_d - v)}$$

2. Accumulated Past Errors - In this error type a Ki term is used, this error sums the speed errors from the beginning till the current time step. This gain is implemented to set the acceleration, based on the history of errors. The integral controller tries to eliminate the steady state errors

$$\boxed{K_I \int_0^t (v_d - v)dt}$$

3. Overshoot Error – In this error a Kd term is used which dampens the overshoot caused by the integration term.

$$\boxed{K_D \frac{d(v_d - v)}{dt}}$$

To control the vehicle then the acceleration output from the PID controller which is “u” is converted into throttle and break commands.

If the value of u comes out to be positive that will correspond to the throttle value and if the value of u comes out to be negative that will correspond to the brake value. In this way there won't be a throttle and brake command simultaneously.

Lateral Control

For implementing the Lateral control, the Stanley control is used. The theory for this section is the same as the one already explained in Section 6.1. This controller also used the kinematic bicycle model to get the steering value for the vehicle.
Using the given reference points from the list, the points are then used to calculate the cross track and the heading error

1. Heading Error – This error is the difference between the current vehicle heading and the reference path heading direction at the reference point along the path. It is a measure of how the vehicle is aligned with the direction of the desired path.
   Unlike the MATLAB example here the reference point is taken to be the front axle. This can be seen in Fig 7.14,

   Ψ is used to represent the relative heading angle of the vehicle with respect to the path line. The front wheel velocity is denoted by $V_{f.}$ The steering angle measured from the heading direction is denoted by δ.

   In this example a line segment is used as the reference path, shown as a solid black line Fig 7.14. A dashed black line that is parallel to the path but runs through the center of the front axle is also visible. For the purposes of lateral control, heading is defined relative to the current path line segment.

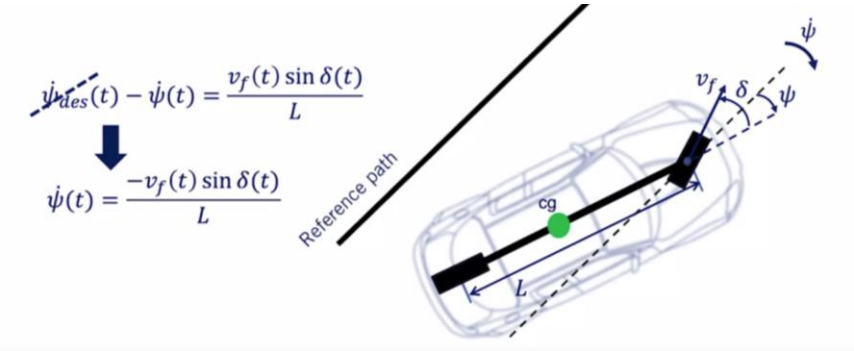

Fig 7.14 Vehicle heading error

Ψ dot is change in desired path with respect to time, which is taken to be zero for a straight aligned path as the reference heading is not time-varying for a straight line and is in fact equal to zero, thus heading is redefined relative to the current path direction.

2. Cross Track Error- This error is the distance between the reference point on the vehicle and the closest point on the desired path. It shows how close the vehicle's position is to the desired position along the path.
   As seen in Fig 7.15, the line from the vehicle reference point to the path reference point is perpendicular to the path. It can be seen that as the velocity of the vehicle increases, the value and rate of change of the cross-track error, this indicates that smaller steering angles are needed to keep cross track errors low.

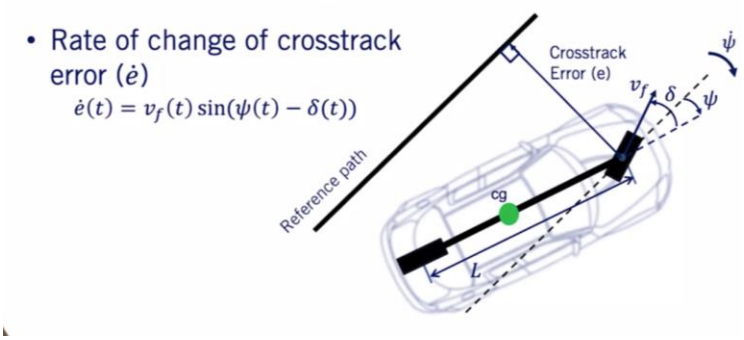

Fig 7.15 Vehicle cross track error

For this example, Stanley control is used in which certain heading control measures are followed.

1. To eliminate heading error, the steering angle δ is directly set equal to the heading angle Ψ as shown here -

$$\delta(t) = \psi(t) \quad \text{- (7.4)}$$

2. To eliminate the cross-track error, a proportional control is added, whose gain is proportional to the inverse of the forward velocity. Then after passing it through an inverse tan function, the steering angle is then capped between a minimum and maximum. Equation 7.5 gives the relation between cross track error and steering.
   K is the proportional gain value, e(t) is cross-track error and vf is forward velocity.
   δ is the steering angle and is capped between a minimum and maximum value.

$$\delta(t) = \tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right) \quad \text{- (7.5)}$$

The combined steering law is given as follows

$$\delta(t) = \psi(t) + \tan^{-1}\left(\frac{ke(t)}{v_f(t)}\right), \quad \delta(t) \in [\delta_{min}, \delta_{max}] \quad \text{- (7.6)}$$

It is important that both heading error and cross track error, converge to 0 for the car to closely follow the path.

Working of the Python Script

A controller_2D class is defined in which first the initialization of all the variables is done. These variables include:

TABLE XIII Variables names with their function

Variable	Variable function
current_x	Stores the current x coordinate
current_y	Stores the current y coordinate
current_yaw	Stores the current heading, given by the IMU
current_speed	Stores the current speed
desired_speed	Contains the desired speed of the vehicle at a way point
set_throttle	Sets the throttle value
set_brake	Sets the braking value
set_steer	Sets the steering value
set_waypoints	Imports the txt file with all the waypoints

The output from both the lateral and longitudinal controller is used to navigate the vehicle in the map. The track is a loop shown in Fig 7.16.

Fig 7.16 Full map of the Carla test environment

The vehicle is to be moved from the starting point to the end point. The way points (Fig 7.17) include the x, y coordinate as well as the desired velocity that the vehicle should have.

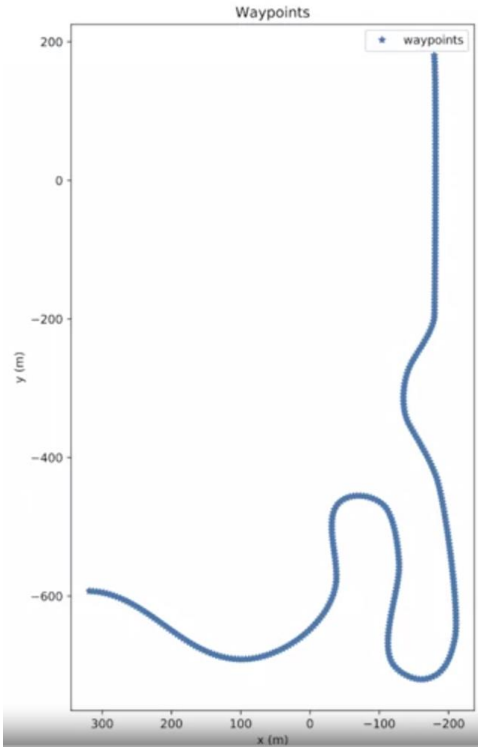

Fig 7.17 Preset Waypoints set in the map

The whole controller can be divided into several smaller functions. Table XIV gives a brief explanation of what each function does

TABLE XIV List of functions and their brief function

Function Name	Description
update_controls	Function used to implement the controller. The input and output variables are described in the following tables. The CARLA client script will first update the state feedback from the CARLA simulator, then update the control outputs using this function and finally send the control outputs to the CARLA simulator.
update_desired_speed	Calculates the desired speed for the controller to track during the current controller iteration.
update_values	Updates latest state feedback from the CARLA simulator.
update_waypoints	Updates the current set of waypoints for the controller to navigate.
get_commands	The CARLA simulator will receive the output commands through this function.
set_brake	Sets the brake command for the CARLA simulator server. Bounds the value between 0 and 1 (0% to 100%)
set_steer	Sets the steer command for the CARLA simulator server. Bounds the value between -1.22 to 1.22 radians
set_throttle	Sets the throttle command for the CARLA simulator server. Bounds the value between 0 and 1 (0% to 100%)

The working of the python script in detail is as follows :

As the vehicle moves on the road, the values of the current x,y, yaw and speed change, thus the update_values() function is called to update these values at every time step.
The update_desired_speed() function transverses through the complete set of

waypoints in the text file and calculates the least distance from the current x,y to the points in the list. This gives us the index of the closest point with respect to the vehicle. The desired speed associated with the closes point is then updated in the variable desired_speed.
The update_controls() function then combines all the functions together. The Proportional, Integral and Differential gain values are initialized, all 3 types of errors in the Longitudinal control sections are defined here, namely the speed error, the integral and differential error. The PID equation is then applied in the code to calculate the acceleration input value . If the value comes out to be positive the value is assigned to the set_throttle variable and if the value comes out to be negative it is assigned to the set_brake variable.
Then for the steering control the heading and cross track errors are defined. The heading error calculated is the current yaw minus the desired yaw of the way point. Similarly, the cross-track error is defined as the minimum distance between the reference point in the vehicle and the closest waypoint. The steering angle is then calculated from these. The maximum steering angle is set to be 70 degrees and if the value of steering goes above 70 then 70 will be used or below -70 then -70 will be used. The update_desired_speed() function is also called within this function to update the desired speed as the waypoints are reached. Once the values of steering, throttle and braking are defined the respective functions are called and the values are updated using the set_steer(), set_throttle() and set_brake() function.

The simulation is then run with the above code, but Kp, Ki, and Kd values have yet to be determined. The parameters were thus tuned by repeated simulation of the vehicle.

Fig 7.18 shows the vehicle launched in the CARLA racetrack environment.

Fig 7.18 Vehicle in racetrack environment

CARLA SIMULATION RESULTS

PID Tuning for the Longitudinal control

To get a set of optimized parameters several simulations were run with different values of Kp, Ki and Kd. To understand the individual contribution of each gain, each of them were increased to a large value while the others were remained to be low. Using this the significance of each parameter was known. Three cases are discussed below with each case corresponds to a specific parameter of the PID controller

CASE I: Proportional Gain (Kp) set high

In this case the Kp value was set at 100 and Ki and Kd values were set at 1. The simulation was then run to see the results. It was expected to see that the system would try to reach the desired value faster, but it might make the system unstable too.

As seen in Fig 7.19, the blue line represents the reference speed that the vehicle should have at a way point and the orange line represents the actual vehicle speed at those specific waypoints. The green dotted line represent the speed threshold band at each of the waypoint. A waypoint is considered successful if the car trajectory is within both thresholds.

It can be observed that the speed of the vehicle oscillates around the reference speed and does not come to a steady state. During change of speed on turns the vehicle velocity overshoots the reference value quickly. In this test only about 88% of the way points were completed

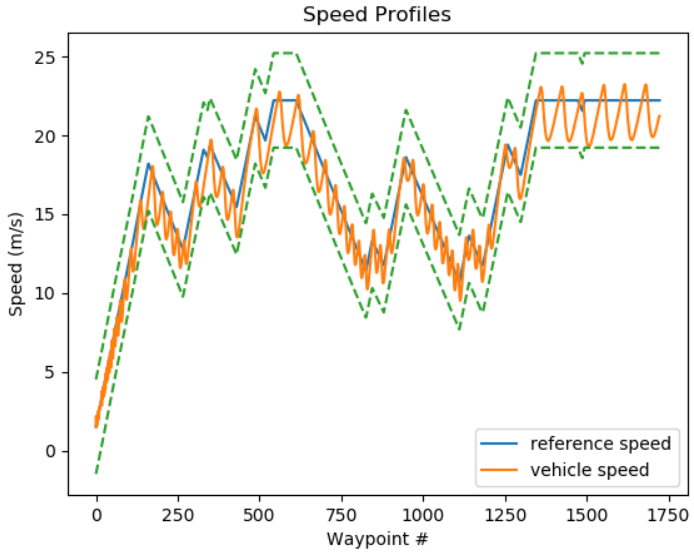

Fig 7.19 Speed profile of the vehicle in CASE I

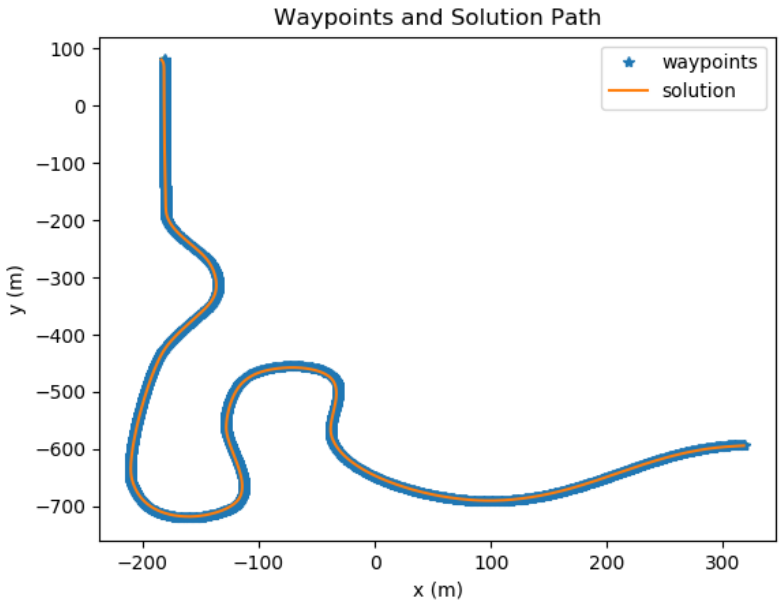

Fig 7.20 Even though the vehicle velocity oscillates a lot it remains inside the speed threshold.

CASE II: Integral Gain (Ki) set high

In this case the Ki value was set at 100 and Kp and Kd values were set at 1. The simulation was then run to see the results. It was expected to see that the system would overshoot very fast. That is having a high Integral value increases the responsiveness of the system

As it can be seen in the Fig 7.21, the blue line represents the reference speed that the vehicle should have at a way point and the orange line represents the actual vehicle speed at those specific waypoints. The green dotted line represents the speed threshold band at each of the waypoint. A waypoint is considered successful if the car trajectory is within both thresholds.

It can be observed that the speed of the vehicle overshoots and then it tries to reach back to the reference speed. During the simulation also, the vehicle's rate of change of speed was very aggressive, it would sometimes go from 8 km/hr to 60 km/hr in a matter of mere seconds and vice versa. Even though the integral value is used to get a steady state error, because of the high value of Ki, the accumulated error keeps on adding up with time and it makes the vehicle very unsafe.
In this test only about 28% of the way points were completed.

Fig 7.21 Speed profile of the vehicle in CASE II

It can also be seen in Fig 7.22 that the vehicle went out of the racetrack, and collided with the walls.

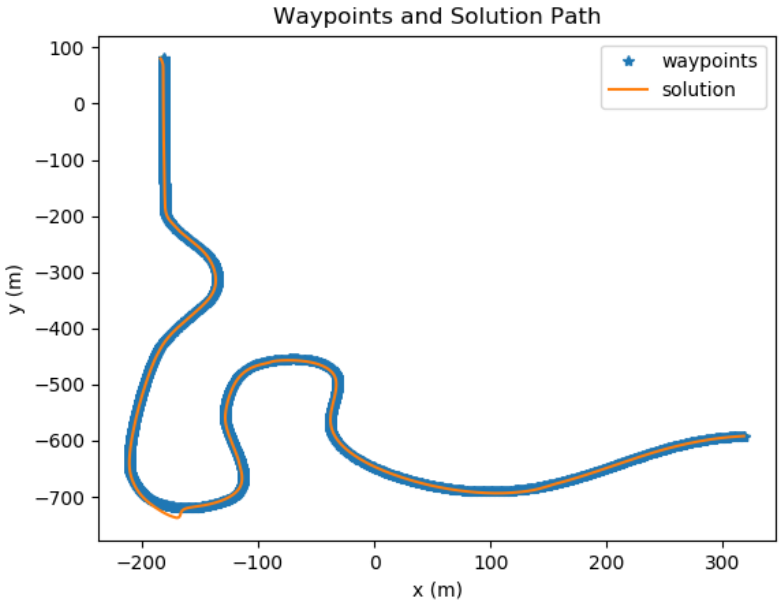

Fig 7.22 Waypoints and solution path

CASE III: Differential Gain (Kd) set high

In this case the Kd value was set at 100 and Kp and Ki values were set at 1. The simulation was then run to see the results. It was expected to see that the system would become very slow as it is now very sensitive to the inside noise. The system is also expected to become slow to react to the changes in speed.

As it can be seen in the Fig 7.23, the blue line represents the reference speed that the vehicle should have at a way point and the orange line represents the actual vehicle speed at those specific waypoints. The green dotted line represent the speed threshold band at each of the waypoint. A waypoint is considered successful if the car trajectory is within both thresholds.

It can be observed that the system was the slowest of the above two. The rate of change of the vehicles speed to match the reference speed was quite low. And if suddenly the reference speed changed the controller was very slow to react to the change in the system. It takes a long time for the controller to reach the reference speed. Even when it becomes close to the reference speed and the reference speed incur a sharp or sudden change, the controller takes so long to reach the desired speed. This is particularly unsafe for a vehicle as, the vehicle should quickly respond to sudden changes in its surroundings. Only about 33% of the way points were completed.

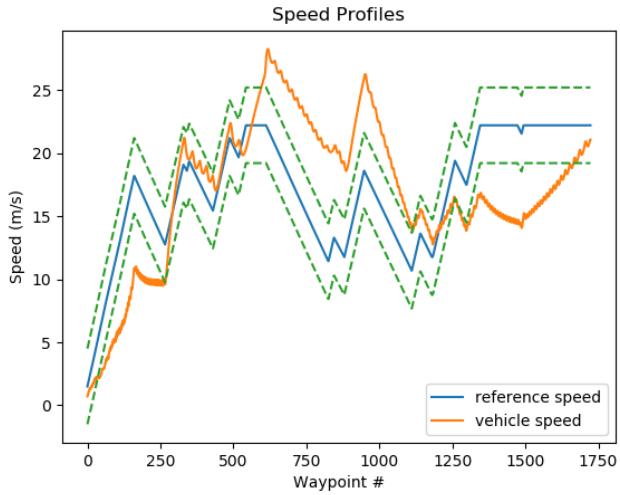

Fig 7.23 Speed profile of the vehicle in CASE III

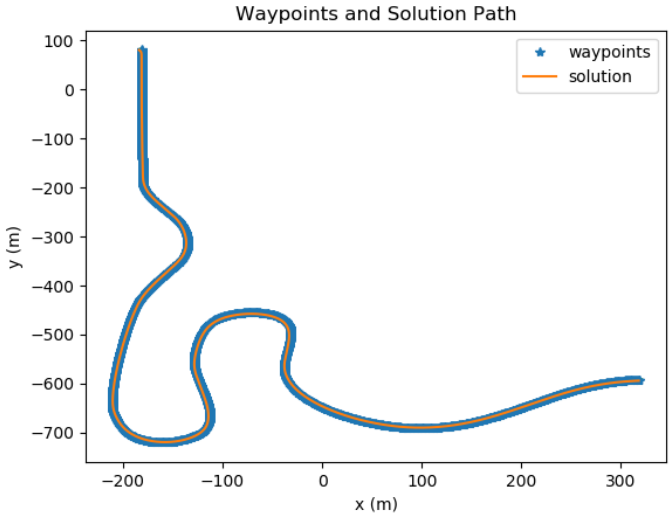

Fig 7.24 Waypoints and solution path

CASE IV: All Gains were optimized

In case IV, Kp was set as 1, Ki - 0.2 and Kd - 0.01. After more than 30 simulation tests, these values were found. System is expected to follow the reference path very closely.

In Fig 7.25, the blue line represents the reference speed that the vehicle should have at a way point and orange line represents the actual speed at those specific waypoints. Green dotted line is the speed threshold band at each of the waypoint. A waypoint is considered successful if the car trajectory is within both thresholds. It can be observed that the system was the most accurate. The vehicle was moving exactly like that of the reference speed. There is no overshoot or lag in this configuration, and this is the best result obtained from this controller. In this test 100% of the way points were completed.

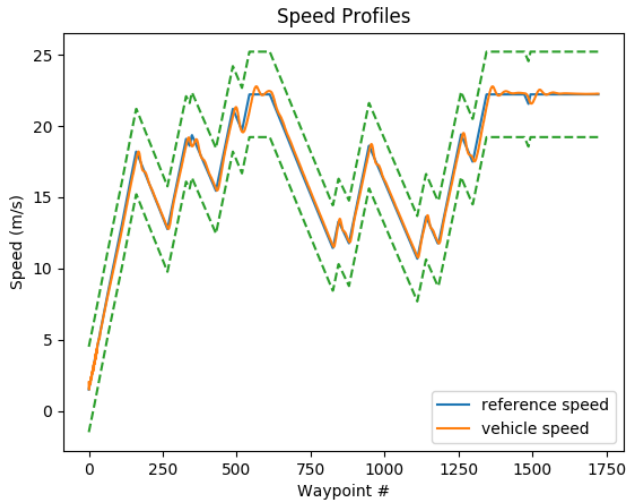

Fig 7.25 Speed profile of the vehicle in CASE IV

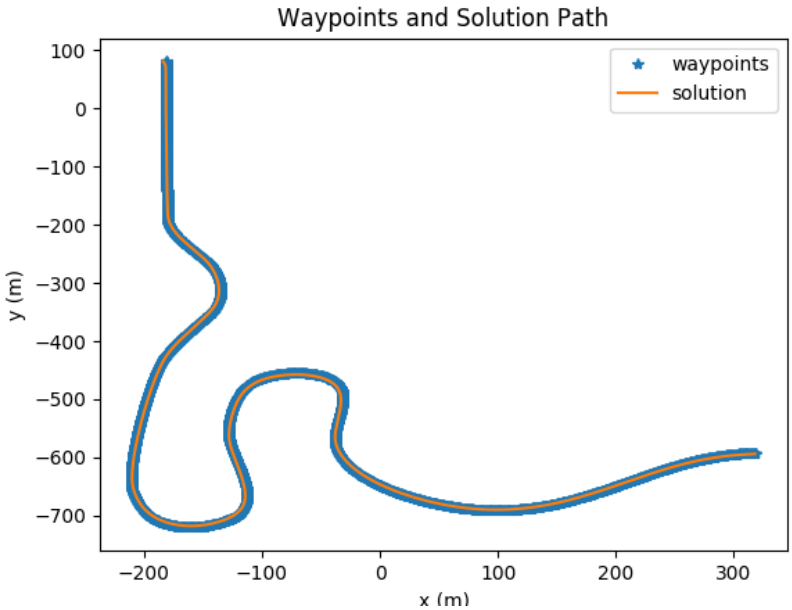

Fig 7.26 Waypoints and solution path

Throttle Value Comparison

The graphs above give the speed profile which is the result of the acceleration that is applied to the car, but the output of the PID controller is the acceleration/ brake value. To simplify the code, braking value was set to be 0, that is if the PID value gave a positive value it would contribute to throttle and if a negative value were to come, the throttle was set to 0.

Case I: Kp value set 100
In this case the throttle was applied momentarily to reach the desired speed quickly and as soon as the speed was achieved, the throttle was let go. In Fig 7.27 the X axis represent the time frame in seconds and Y axis represents the throttle value from 0 to 1.

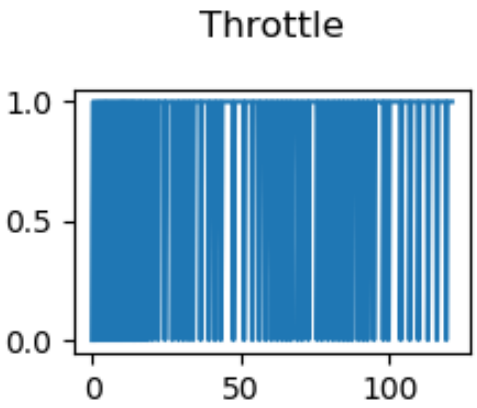

Fig 7.27 Throttle graph for case I

Case II: Ki value set 100
In this case the throttle was applied for a long time to quickly reach the speed but this resulted in big overshoots between the actual speed and the desired speed. Since the braking was set to 0, in this case the throttle is let go for long time intervals. In Fig 7.28 the X axis represent the time frame in seconds and the Y axis represents the throttle value from 0 to 1.

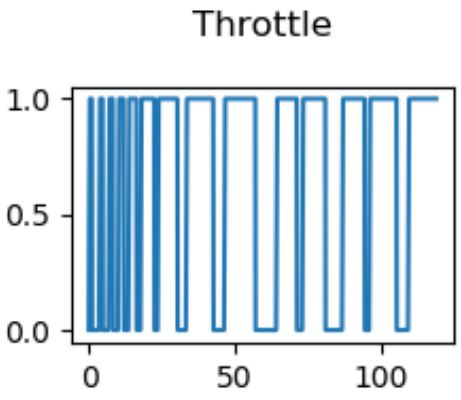

Fig 7.28 Throttle graph for case II

Case III: Kd value set 100

In this case the throttle was applied momentarily, and also not to its maximum. The controller tried to keep its rate of change of speed to a very low value. Only when once the vehicle gained speed, the controller the applied the full throttle. There were no instances where the throttle was constantly pressed for a substantial amount of time. This can be concluded from the time frame on the X axis. This case took the most time, as the vehicle travelled very slowly. In Fig 7.29 the X axis represent the time frame in seconds and the Y axis represent the throttle value from 0 to 1.

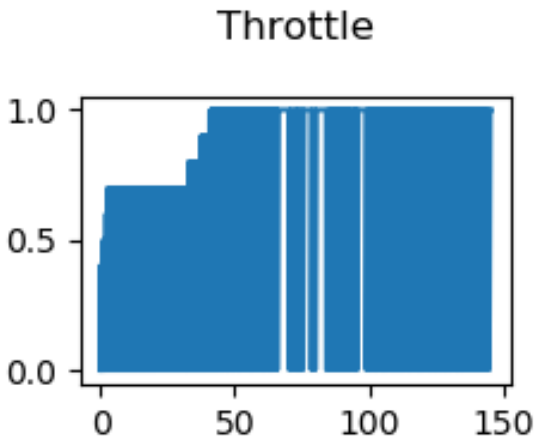

Fig 7.29 Throttle graph for case III

Case IV: PID values optimized

In this case the throttle was applied throughout, and not once after the initial 20-30 seconds did the throttle come to 0. The throttle values in this case had a very smooth transition and there are no cases of sudden pressing of the throttle value . The controller tried to keep its rate of change of speed to a very comfortable value. As seen in Fig 7.30 the X axis represents the time frame in seconds and Y axis represents the throttle value from 0 to 1.

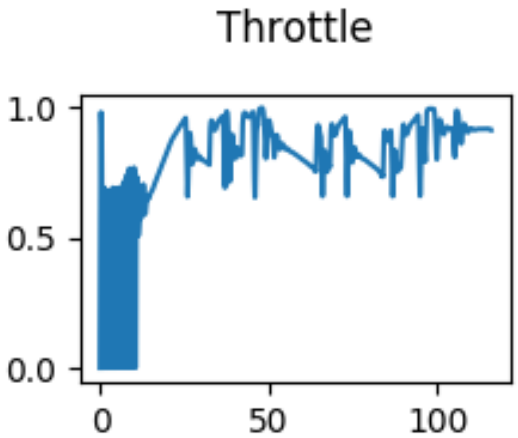

Fig 7.30 Throttle graph for case IV

Steering Results

The steering values were also plotted for each case above. The results for the steering value for almost each case were quite similar, which is understandable also. As after a few way points have been transversed the lateral controller minimizes the errors and gets a fair steering value. Only during turns the acceleration or the speed is too high this can result in the vehicle to drift out of the track. This had happened in case II where the vehicle crashed into the wall during turning

The steering values are set between 1 and -1, where +1 represents 35 degree turn. The plots for all 4 cases are plotted below. The X axis represents the time frame in seconds and the Y axis represents the steering value.

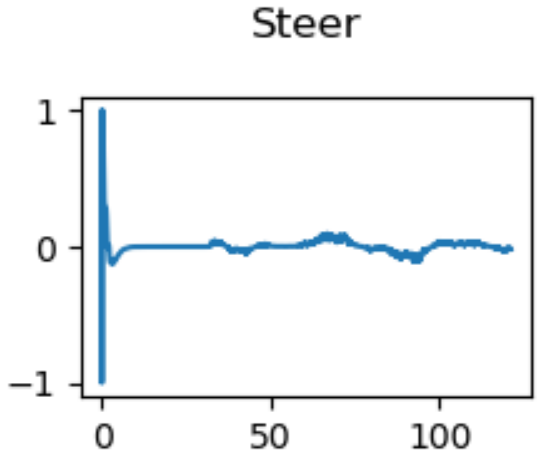

Fig 7.31(a) Steering value for Case I
Kp value- 100

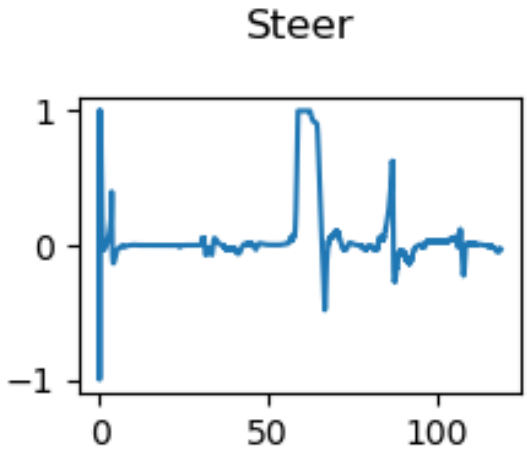

Fig 7.31(b) Steering value for Case II
Ki Value - 100

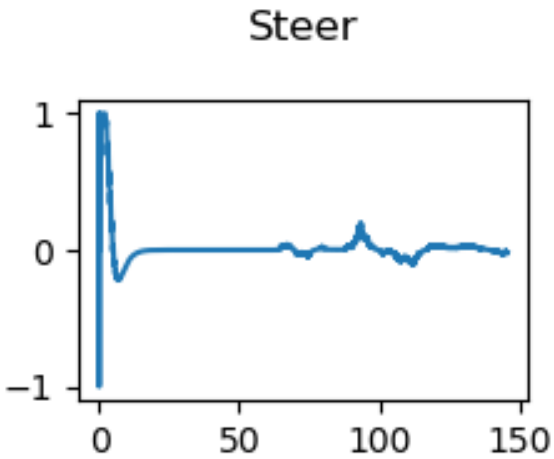

Fig 7.31(c) Steering value for Case III
Kd Value- 100

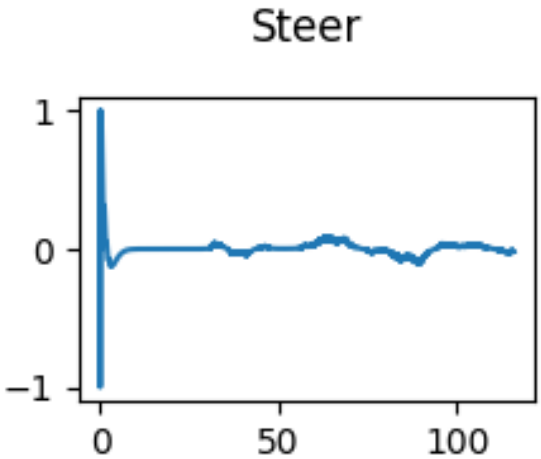

Fig 7.31(d) Steering value for Case IV
Kp, Ki, Kd optimized values

Fig 7.31 Steering values

In the figures it is observed that, Fig 7.31(a) case I steering resemble the closest to the optimized steering graph. In Fig 7.31(b) case II the steering suddenly increased from 0 to 70 degrees which resulted in the car drifting outside of the track. The overall steering in Fig 7.31(b) case II is not steady and can cause uncomfortable feeling to the passengers. The Fig 7.31(c) case III, the graph might appear acceptable, but the time frame is the highest of all the four. Also, it experiences certain noise in the later part of the drive. The Fig 7.31(d) is the steering for the optimized PID gains where other than the initial correction the whole ride remains smooth.

The vehicle was thus able to move from the starting point to the end point of the map by taking the waypoints as reference while simultaneously correcting for the steering and the throttle values (Fig 7.32). The waypoints generation in this case was static but later it can be dynamic where the GPS sensor might give the X and Y coordinate for the vehicle to follow, while also taking sensor data from a camera to avoid obstacles while steering through the given waypoints.

Fig 7.32 Vehicle moving using waypoints

Chapter 8: Conclusions and Results

8.1 Conclusion

8.1.1 Motion planning using visual perception

Two different lane detection models – Linear lane detection and Parabolic lane detection were implemented. To get the lane markers in Linear model, techniques like Color masking, Canny edge detection, Hough line transform were used. The parabolic model used 2D-curve fitting (on filtered/transformed frames) using RANSAC algorithm and generated boundaries.

The detections were compared, and parabolic lane detection was slightly superior than the linear one. For developing the lane following system, steering angles were successfully generated on both lane detection algorithms.

8.1.2 Collision Warning System

The MATLAB ADAS toolbox and its ability to track and detect objects is presented in this report. The detector was used on self-made videos using a monocular camera. The overall result of cars, vehicles and pedestrians is quite great.
A collision warning system, which is used for acceleration and braking was successfully implemented using MATLAB's ADAS toolbox. This can later be integrated with ROS on the E2O vehicle.

But the ACF algorithm was not perfect, there were some limitations with it.

- There were some inaccurate detection of objects
- In a traffic jam with many cars or a crowd with many pedestrians it might miscount or give incorrect results
- Lighting conditions are a constraint, there were no tests conducted in rainy weather thus it can have bad detections in the same

The results can sure be improved by fusing other sensor data with this one.

8.1.3 Path planning and miscellaneous functions

Path planning techniques were analyzed. A Bicycle kinematic model was implemented to obtain the steering angle for a simulated vehicle. Simple 2D techniques of path planning, trajectory generation, and vehicle control were implemented on a simple simulation. The ROS Toolbox was also explored to form an interface between MATLAB and ROS. We also implemented Traffic signal detection using CNN. The dataset can be trained on the types of road signs seen in the campus and the class Ids can be changed accordingly.
An 2D occupancy grid was also generated to make a vehicle cost map which would be helpful for path planning.
On CARLA simulation, static waypoint following was also implemented using appropriate steering and throttle values.

8.2 Future Work

All these algorithms have been tested on real-world videos. Due to the constraint of working remotely, they have not implemented this in the car. All this could be integrated into the car. On achieving successful functioning, the car could run driverless in IIT Delhi campus and also be responsive to the immediate environment.

References

[1] https://in.mathworks.com/help/images/apply-gaussian-smoothing-filters-to-images.html
[2] G. T. Shrivakshan and Dr. C. Chandrasekar, "A Comparison of various Edge Detection Techniques used in Image Processing", International Journal of Computer Science Issues, Vol. 9, Issue 5, No 1, September 2012.
[3] He Mao and Mei Xie, "Lane detection based on hough transform and endpoints classification," 2012 International Conference on Wavelet Active Media Technology and Information Processing (ICWAMTIP), Chengdu, 2012, pp. 125-127, doi: 10.1109/ICWAMTIP.2012.6413455.
[4] https://in.mathworks.com/help/vision/ug/single-camera-calibrator-app.html
[5] J. Wang, F. Gu, C. Zhang and G. Zhang, "Lane boundary detection based on parabola model," The 2010 IEEE International Conference on Information and Automation, Harbin, 2010, pp. 1729-1734, doi: 10.1109/ICINFA.2010.5512219.
[6] https://in.mathworks.com/help/driving/ref/segmentlanemarkerridge.html
[7] https://in.mathworks.com/help/driving/ref/paraboliclaneboundary.html
[8] Ding, Y., Xu, Z., Zhang, Y. et al. Fast lane detection based on bird's eye view and improved random sample consensus algorithm. Multimed Tools Appl 76, 22979–22998 (2017). https://doi.org/10.1007/s11042-016-4184-6
[9] @inproceedings{Derpanis2005OverviewOT, title={Overview of the RANSAC Algorithm}, author={Konstantinos G. Derpanis}, year={2005}}
[10] Birds Eye Vie Transformation-International Journal of Scientific & Engineering Research, Volume 3, Issue 5, May-2012 1 ISSN 2229-5518
[11] http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/
[12] T. Kokul, A. Ramanan and U. A. J. Pinidiyaarachchi, "Online multi-person tracking-by-detection method using ACF and particle filter," 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, 2015, pp. 529-536, doi: 10.1109/IntelCIS.2015.7397272.
[13] Y. Byeon and K. Kwak, "A Performance Comparison of Pedestrian Detection Using Faster RCNN and ACF," 2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), Hamamatsu, 2017, pp. 858-863, doi: 10.1109/IIAI-AAI.2017.196
[14] https://in.mathworks.com/help/control/ug/kalman-filtering.html
[15] Rajaram, R.N., Ohn-Bar, E. and Trivedi, M.M., 2016, November. A study of vehicle detector generalization on US highway. In 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC) (pp. 277-282). IEEE.
[16] P. Corke, "Integrating ROS and MATLAB [ROS Topics]," in IEEE Robotics & Automation Magazine, vol. 22, no. 2, pp. 18-20, June 2015, doi: 10.1109/MRA.2015.2418513.
[17] https://in.mathworks.com/help/ros/ug/get-started-with-ros.html#:~:text=View%20MATLAB%20Command,and%20receive%20data%20between%20them.
[18] https://in.mathworks.com/help/driving/examples/automated-parking-valet.html
[19] https://in.mathworks.com/help/driving/ug/coordinate-systems.html
[20] Abbadi, A. and Matousek, R., 2014. Path Planning Implementation Using MATLAB. Technical Computing Bratislava, pp.1-5.
[21] https://in.mathworks.com/help/driving/ref/pathplannerrrt.html
[22] https://in.mathworks.com/help/driving/ref/smoothpathspline.html
[23] Floater, Michael S. "On the Deviation of a Parametric Cubic Spline Interpolant from Its Data Polygon." Computer Aided Geometric Design. Vol. 25, Number 3, 2008, pp. 148–156.
[24] https://medium.com/@dingyan7361/simple-understanding-of-kinematic-bicycle-model-81cac6420357
[25] https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8
[26] https://towardsdatascience.com/traffic-sign-detection-using-convolutional-neural-network-660fb32fe90e
[27] Shustanov, A. and Yakimov, P., 2017. CNN design for real-time traffic sign recognition. Procedia engineering, 201, pp.718-725.
[28] https://in.mathworks.com/help/driving/examples/create-occupancy-grid-using-monocular-camera-sensor.html
[29] Brostow, Gabriel J., Julien Fauqueur, and Roberto Cipolla. "Semantic Object Classes in Video: A high-definition ground truth database." Pattern Recognition Letters. Vol. 30, Issue 2, 2009, pp. 88-97.
[30]https://in.mathworks.com/help/driving/ref/longitudinalcontrollerstanley.html#mw_e0c62d0f-29b0-4b4e-b27e-04180cae6694

[31] A. B. P. Dollar.P, "Fast Feature Pyramids for Object Detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 8, pp. 1532-1545, 2014.
[32] W.-C. Chang and C.-W. Cho, "Online Boosting for Vehicle Detection," IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol. 40, no. 3, pp. 892-902, 2010.
[33] Khalifa, O.O., Islam, R., Am Assidi, A., Abdullah, A.H. and Khan, S., 2011. Vision based road lane detection system for vehicles guidance. Aust. J. Basic Appl. Sci, 5(5), pp.728-738.
[34] Seo, Y.W. and Rajkumar, R., 2013, May. Use of a monocular camera to analyze a ground vehicle's lateral movements for reliable autonomous city driving. In Proceedings of IEEE IROS Workshop on Planning, Perception and Navigation for Intelligent Vehicles (pp. 197-203).
[35] Aytekin, B. and Altuğ, E., 2010, October. Increasing driving safety with a multiple vehicle detection and tracking system using ongoing vehicle shadow information. In 2010 IEEE International Conference on Systems, Man and Cybernetics (pp. 3650-3656). IEEE.
[36] Gilmore, E.T., Ugbome, C. and Kim, C., 2011. An IR-based pedestrian detection system implemented with matlab-equipped laptop and low-cost microcontroller. International Journal of Computer Science & Information Technology, 3(5), p.79
[37] Zhang, L., Lin, L., Liang, X. and He, K., 2016, October. Is faster R-CNN doing well for pedestrian detection?. In European conference on computer vision (pp. 443-457). Springer, Cham
[38] P. I. Corke, Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Berlin Heidelberg, Germany: Springer-Verlag, 2011
[39] P. Corke. Robotics toolbox for MATLAB. [Online]. Available: http://www.petercorke.com/robot
[40] https://www.mathworks.com/products/robotics.html

Appendix A

CODE FOR LINEAR LANE DETECTION AND STEERING

```
%-------Initialize code--------
clc
clear all
close all

%------Reading the mp4 File-----
Input_vid  = VideoReader('car_4.mp4');

%----Defining the output File----
Out_vid = VideoWriter('Result_sample_car_4');
Out_vid.FrameRate = 30;
open(Out_vid);

while hasFrame(Input_vid)

    %------------------Input stream of frames----------
    frame = readFrame(Input_vid);
    %   imshow(frame);

    frame = imgaussfilt3(frame);
    %    figure('Name','Gauss blurred Image'), imshow(frame);

    %-------------- White Mask--------------
    %-------Define optimal range for Hue, Saturation and Value-------------
    HueMin = 170;
    HueMax = 250;

    SatMin = 170;
    SatMax = 250;

    ValMin = 170;
    ValMax = 250;

    %-----------Mask based on selected thresholds-----------


White_masked_frame=((frame(:,:,1)>=HueMin)|(frame(:,:,1)<=HueMax))&(frame(:,:,2)
>=SatMin)&...
    (frame(:,:,2)<=SatMax)&(frame(:,:,3)>=ValMin)&(frame(:,:,3)<=ValMax);

    % figure('binary_masked'), imshow(White_masked_frame);

    %-----------------Edge Detection--------------------------------

    New_frame = edge(White_masked_frame, 'canny');
    New_frame = bwareaopen(New_frame,50);

    % imshow(New_frame);
```

```
%---------Defining a Region of Interest for Left lane---------

x =[200,700,600,0];
y = [300,300,1000,1000];
left_roi = roipoly(New_frame, x, y);
[X , Y] = size(left_roi);
for i = 1:X
    for j = 1:Y
        if left_roi(i,j) == 1
            left_roi_frame(i,j) = New_frame(i,j);
        else
            left_roi_frame(i,j) = 0;
        end
    end
end

%  imshow(left_roi_frame);

%---------Defining a Region of Interest for Right lane----------

x =[900,1100,1500,1200];
y = [100,100,1000,1000];
right_roi = roipoly(New_frame, x, y);
[X , Y] = size(right_roi);
for i = 1:X
    for j = 1:Y
        if right_roi(i,j) == 1
            right_roi_frame(i,j) = New_frame(i,j);
        else
            right_roi_frame(i,j) = 0;
        end
    end
end

%  imshow(right_roi_frame);

%------Hough Tansform on edge detected images to generate lines--------

[Left_H_matrix,left_theta,left_rho] = hough(left_roi_frame);
[Right_H_matrix,right_theta,right_rho] = hough(right_roi_frame);

%-----------------Get 3 Peaks from each Hough Transform----------------

Left_peaks_temp = houghpeaks(Left_H_matrix,3,'threshold',3);
Right_peaks_temp = houghpeaks(Right_H_matrix,3,'threshold',3);

%-------If peaks not detected, use previous value----------------------
if sum(size(Left_peaks_temp))==5
    Left_peaks=Left_peaks_temp;
end

if sum(size(Right_peaks_temp))==5
    Right_peaks=Right_peaks_temp;
end

%--------------Extracting Lines from Detected Hough Peaks--------------
```

```
    left_lane_lines_temp =
houghlines(left_roi_frame,left_theta,left_rho,Left_peaks,'FillGap',3000,'MinLeng
th',20);

    right_lane_lines_temp =
houghlines(right_roi_frame,right_theta,right_rho,Right_peaks,'FillGap',3000,'Min
Length',20);

    if sum(size(left_lane_lines_temp))==4
        left_lane_lines = left_lane_lines_temp;
    end

    if sum(size(right_lane_lines_temp))==4
        right_lane_lines = right_lane_lines_temp;
    end

    %     max_len = 0;
    %     for k = 1:2
    %        xy = [right_lane_lines(k).point1; right_lane_lines(k).point2];
    %        plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
    %     end
    %     hold off

    %-----------------Best fitting lane lines----------------

    leftp1 = [left_lane_lines(1).point1; left_lane_lines(1).point2];
    leftp2 = [left_lane_lines(2).point1; left_lane_lines(2).point2];

    rightp1 = [right_lane_lines(1).point1; right_lane_lines(1).point2];
    rightp2 = [right_lane_lines(2).point1; right_lane_lines(2).point2];

    if leftp1(1,1) < leftp2(1,1)
        left_lane(1,:) = leftp1(1,:);
    else
        left_lane(1,:) = leftp2(1,:);
    end

    if leftp1(2,2) < leftp2(2,2)
        left_lane(2,:) = leftp1(2,:);
    else
        left_lane(2,:) = leftp2(2,:);
    end

    if rightp1(1,2) < rightp2(1,2)
        right_lane(1,:) = rightp1(1,:);
    else
        right_lane(1,:) = rightp2(1,:);
    end

    if rightp1(2,1) > rightp2(2,2)
        right_lane(2,:) = rightp1(2,:);
    else
        right_lane(2,:) = rightp2(2,:);
    end

    %---------------Finding lane slopes and extrapolating---------------

    %Calculating slope using end points
    slopeL = (left_lane(2,2)-left_lane(1,2))/(left_lane(2,1)-left_lane(1,1));
```

```
    slopeR = (right_lane(2,2)-right_lane(1,2))/(right_lane(2,1)-
right_lane(1,1));

    % Specifying X coordinates and then finding Y coordinates using slope

    %Start from left edge
    Left_lane_x_left = 1;
    Left_lane_x_right = 350;
    %Finding corresponsding y coordinates
    Left_lane_y_left = slopeL * (Left_lane_x_left - left_lane(1,1)) +
left_lane(1,2);
    Left_lane_y_right = slopeL * (Left_lane_x_right - left_lane(2,1)) +
left_lane(2,2);

    % Start from right edge
    Right_lane_x_left = 950;
    Right_lane_x_right = 1300;
    %Finding corresponsding y coordinates
    Right_lane_y_left = slopeR * (Right_lane_x_left - right_lane(1,1)) +
right_lane(1,2);
    Right_lane_y_right = slopeR * (Right_lane_x_right - right_lane(2,1)) +
right_lane(2,2);

    %------Highlighting the safe region between lanes-------
    points = [Left_lane_x_left Left_lane_y_left; Left_lane_x_right
Left_lane_y_right ;Right_lane_x_left Right_lane_y_left; Right_lane_x_right
Right_lane_y_right ];
    number = [1 2 3 4];

    %-----------------Plot the result on original frame--------------------

    %figure('Name','Final Output')
    imshow(frame);
    [height, width, dim] = size(frame);
    hold on
    % frame centre marker
    plot(width/2,height/2,'r+', 'MarkerSize', 50);
    ww = ((left_lane(2,1)+left_lane(1,1))/2+(right_lane(1,1)+
right_lane(2,1))/2)/2 - width/2;
    % deviation marker
    plot(((left_lane(2,1)+left_lane(1,1))/2+(right_lane(1,1)+
right_lane(2,1))/2)/2,height/2,'b+', 'MarkerSize', 50);
    % calculating steering angle
    steering = atand(ww/(height/2));
    print_steer = ['Steer: ' num2str(steering,'%0.2f')];

    % plotting lane lines
    plot([Left_lane_x_left, Left_lane_x_right], [Left_lane_y_left,
Left_lane_y_right], 'LineWidth',8,'Color','red');
    plot([Right_lane_x_left, Right_lane_x_right], [Right_lane_y_left,
Right_lane_y_right], 'LineWidth',8,'Color','red');

    if steering>=-4 && steering<=4
       % print straight
       text(650, 65, 'Straight','horizontalAlignment', 'center',
'Color','red','FontSize',30)
    else
       % print steering angle
```

```
       text(650, 65, print_steer,'horizontalAlignment', 'center',
'Color','red','FontSize',30)
    end
    % plotting trapezoid
    patch('Faces', number, 'Vertices', points,
'FaceColor','blue','Edgecolor','blue','FaceAlpha',0.4)
    hold off

    %------------------Saving frame in output file------------------
    writeVideo(Out_vid,getframe);
end

%-------------------------------Closing-----------------------------------
close(Out_vid)
```

Appendix B

CODE FOR PARABOLIC LANE DETECTION AND STEERING

```
%-------Initialize code--------
clc
clear all
close all

%------Reading the mp4 File-----
Input_vid = VideoReader('car_4_short.mp4');

%----Defining the output File---------------
Out_vid=VideoWriter('Result_parabolic_lane');
Out_vid.FrameRate= 30;
open(Out_vid);

%------------Camera callibration------------------
% intrinsics
% in pixel units
focalLength    = [771.6633, 773.6966];
principalPoint = [622.8061, 379.3357];
imageSize      = [720, 1280];

camIntrinsics = cameraIntrinsics(focalLength, principalPoint, imageSize);

% extrinsics
height = 1.2;    % in meters
pitch  = 15;     % in degrees

sensor = monoCamera(camIntrinsics, height, 'Pitch', pitch);

while hasFrame(Input_vid)

    %------------------Input stream of frames----------
    frame = readFrame(Input_vid);
    % imshow(frame)

    % Define the transformation area in vehicle coordinates
    % in meters
    distAheadOfSensor = 30;  %  xmax
    spaceToleftSide    = 10; %  -xmin
    spaceTorightSide   = 10; %  ymax
    bottomOffset      = 1;   % ymin

    outFrame   = [bottomOffset, distAheadOfSensor, -spaceToleftSide,
spaceTorightSide];
    % width defined and height automatically adjusted
    imageSize = [NaN, 250];
    birdsEyeConfig = birdsEyeView(sensor, outFrame, imageSize);
    birdsEyeImage = transformImage(birdsEyeConfig, frame);
```

```
    % Grayscale conversion
    birdsEyeImage = rgb2gray(birdsEyeImage);
    %figure
    %imshow(birdsEyeImage)

    % Defining Region of Interest
    vehicleROI = outFrame - [-1, 2, -1, 1];
    % For lane identification
    LaneMarkerWidthWorld = 0.25; % 25 centimeters
    laneSensitivity = 0.25;
    % Give candidate lane points
    birdsEyeViewBW = segmentLaneMarkerRidge(birdsEyeImage, birdsEyeConfig,
LaneMarkerWidthWorld,...
        'ROI', vehicleROI, 'Sensitivity', laneSensitivity);

    % Convert the lane points in world coordinates
    [imageX, imageY] = find(birdsEyeViewBW);
    xyBoundaryPoints = imageToVehicle(birdsEyeConfig, [imageY, imageX]);
    maxLaneCount      = 2;
    boundaryWidth = 3*LaneMarkerWidthWorld; % expand boundary width

    % fitting a parabolic lane model
    % using RANSAC algorithm
    [boundaries, boundaryPoints] =
findParabolicLaneBoundaries(xyBoundaryPoints,boundaryWidth, ...
        'MaxNumBoundaries', maxLaneCount, 'validateBoundaryFcn',
@validateBoundaryFcn);

    xOffset    = 0;
    distanceToBoundaries  = boundaries.computeBoundaryModel(xOffset);
    % Find individual lane lines
    leftIndices  = [];
    rightIndices = [];
    minLDistance = min(distanceToBoundaries(distanceToBoundaries>0));
    minRDistance = max(distanceToBoundaries(distanceToBoundaries<=0));
    if ~isempty(minLDistance)
        leftIndices  = distanceToBoundaries == minLDistance;
    end
    if ~isempty(minRDistance)
        rightIndices = distanceToBoundaries == minRDistance;
    end
    leftLane       = boundaries(leftIndices);
    rightLane      = boundaries(rightIndices);
    % selected range of points
    xCoordsWorld = bottomOffset:distAheadOfSensor;

    %-------------Plotting lanes---------------------------
    birdsEyeWithEgoLane = insertLaneBoundary(birdsEyeImage, leftLane ,
birdsEyeConfig, xCoordsWorld, 'Color','Red');
    birdsEyeWithEgoLane = insertLaneBoundary(birdsEyeWithEgoLane, rightLane,
birdsEyeConfig, xCoordsWorld, 'Color','Green');
    % imshow(birdsEyeWithEgoLane);

    % Define a black frame of same size as original
    blackFrame = 255 * zeros(720, 1280, 3, 'uint8');
    hold on;
    % creating pseudo binary image
    blackFrame = insertLaneBoundary(blackFrame, leftLane, sensor, xCoordsWorld,
'Color','White');
```

```
    blackFrame = insertLaneBoundary(blackFrame, rightLane, sensor, xCoordsWorld,
'Color','White');
    % lanes on original image
    frame = insertLaneBoundary(frame, leftLane, sensor, xCoordsWorld,
'Color','Red');
    frame = insertLaneBoundary(frame, rightLane, sensor, xCoordsWorld,
'Color','Green');
    imshow(frame);
    % steering value calculated
    steering = steer_fun(blackFrame);
    %disp(steering);
    print_steer = ['Steer: ' num2str(steering,'%0.2f')];
    text(650, 65, print_steer,'horizontalAlignment', 'center',
'Color','red','FontSize',30)

    hold off
    writeVideo(Out_vid,getframe);
end
close(Out_vid)




%----------------------------------FUNCTIONS-------------------------------------
---------

% Function 1
function isGood = validateBoundaryFcn(params)
% curves with very high curvature unlikely to be lanes
if ~isempty(params)
    a = params(1); % ax^2+bx+c
    % small a = highly curved
    isGood = abs(a) < 0.003;
else
    isGood = false;
end
end

% Function 3 borrowed from linear lane steering
function steering = steer_fun(frame)

        HueMin = 130;
        HueMax = 255;

        SatMin = 130;
        SatMax = 255;

        ValMin = 130;
        ValMax = 255;
```

```
        %-----------Mask based on selected thresholds-----------

White_masked_frame=((frame(:,:,1)>=HueMin)|(frame(:,:,1)<=HueMax))&(frame(:,:,2)
>=SatMin)&...
        (frame(:,:,2)<=SatMax)&(frame(:,:,3)>=ValMin)&(frame(:,:,3)<=ValMax);

         New_frame = edge(White_masked_frame, 'canny');

        x =[0,600,600,0];
        y = [100,100,1000,1000];
        left_roi = roipoly(New_frame, x, y);
        [X , Y] = size(left_roi);
        for i = 1:X
            for j = 1:Y
                if left_roi(i,j) == 1
                    left_roi_frame(i,j) = New_frame(i,j);
                else
                    left_roi_frame(i,j) = 0;
                end
            end
        end

        x =[1300,750,750,1300];
        y = [100,100,1000,1000];
        right_roi = roipoly(New_frame, x, y);
        [X , Y] = size(right_roi);
        for i = 1:X
            for j = 1:Y
                if right_roi(i,j) == 1
                    right_roi_frame(i,j) = New_frame(i,j);
                else
                    right_roi_frame(i,j) = 0;
                end
            end
        end

       %-----Hough Tansform on edge detected images to generate lines-----

       [Left_H_matrix,left_theta,left_rho] = hough(left_roi_frame);
       [Right_H_matrix,right_theta,right_rho] = hough(right_roi_frame);

       Left_peaks_temp = houghpeaks(Left_H_matrix,3,'threshold',3);
       Right_peaks_temp = houghpeaks(Right_H_matrix,3,'threshold',3);

       %-------If peaks not detected, use previous value---------------
       if sum(size(Left_peaks_temp))==5
          Left_peaks=Left_peaks_temp;
       else
          Left_peaks=[1939,44 : 1975,46 : 2009,48];
       end
       if sum(size(Right_peaks_temp))==5
          Right_peaks=Right_peaks_temp;
       end

        %------------Extracting Lines from Detected Hough Peaks--------

        left_lane_lines_temp =
houghlines(left_roi_frame,left_theta,left_rho,Left_peaks,'FillGap',3000,'MinLeng
th',20);
```

```
        right_lane_lines_temp =
houghlines(right_roi_frame,right_theta,right_rho,Right_peaks,'FillGap',3000,'Min
Length',20);

        if sum(size(left_lane_lines_temp))==4
            left_lane_lines = left_lane_lines_temp;
        end

        if sum(size(right_lane_lines_temp))==4
            right_lane_lines = right_lane_lines_temp;
        end

        %-----------------Best fitting lane lines----------------

        leftp1 = [left_lane_lines(1).point1; left_lane_lines(1).point2];
        leftp2 = [left_lane_lines(2).point1; left_lane_lines(2).point2];

        rightp1 = [right_lane_lines(1).point1; right_lane_lines(1).point2];
        rightp2 = [right_lane_lines(2).point1; right_lane_lines(2).point2];

        if leftp1(1,1) < leftp2(1,1)
            left_lane(1,:) = leftp1(1,:);
        else
            left_lane(1,:) = leftp2(1,:);
        end

        if leftp1(2,2) < leftp2(2,2)
            left_lane(2,:) = leftp1(2,:);
        else
            left_lane(2,:) = leftp2(2,:);
        end

        if rightp1(1,2) < rightp2(1,2)
            right_lane(1,:) = rightp1(1,:);
        else
            right_lane(1,:) = rightp2(1,:);
        end

        if rightp1(2,1) > rightp2(2,2)
            right_lane(2,:) = rightp1(2,:);
        else
            right_lane(2,:) = rightp2(2,:);
        end

        %-------------Finding lane slopes and extrapolating---------------

        %Calculating slope using end points
        slopeL = (left_lane(2,2)-left_lane(1,2))/(left_lane(2,1)-
left_lane(1,1));
        slopeR = (right_lane(2,2)-right_lane(1,2))/(right_lane(2,1)-
right_lane(1,1));
        %Start from left edge
        Left_lane_x_left = 1;
        Left_lane_x_right = 350;
        Left_lane_y_left = slopeL * (Left_lane_x_left - left_lane(1,1)) +
left_lane(1,2);
        Left_lane_y_right = slopeL * (Left_lane_x_right - left_lane(2,1)) +
left_lane(2,2);
```

```
        % Start from right edge
        Right_lane_x_left = 950;
        Right_lane_x_right = 1300;
        Right_lane_y_left = slopeR * (Right_lane_x_left - right_lane(1,1)) +
right_lane(1,2);
        Right_lane_y_right = slopeR * (Right_lane_x_right - right_lane(2,1)) +
right_lane(2,2);

        [height, width, dim] = size(frame);
        ww = ((left_lane(2,1)+left_lane(1,1))/2+(right_lane(1,1)+
right_lane(2,1))/2)/2 - width/2;
        steering = (atand(ww/(height/2)))/10;
end
```

Appendix C

CODE FOR VEHICLE AND PEDEDTRIAN DETECTION

```
%--------------------Set the camera Parameters----------------------
focal_Length    = [772.6633 775.6966]; % Focal Length
principal_Point = [622.8061 379.3357]; % The Optical Center
image_Size      = [720 1280];          % Image Size
camIntrinsics = cameraIntrinsics(focal_Length, principal_Point,
image_Size);
height = 1.4;    % Height of the camera from the gound
pitch  = 10;     % Pitch value of the camera
%The sensor is defined with the above parameters
sensor = monoCamera(camIntrinsics, height, 'Pitch', pitch);

%The pretrained Aggregate Channel Features, model is loaded
% The full view model is used which is trained on full images of
vehicles
Vehicle_detector = vehicleDetectorACF('full-view');

%The pedestrain detector is loaded with is trained on images of
pedestrians
Ped_detector = peopleDetectorACF();

% Average width of pedestrians and vehicles is defined
vehicle_Width = [1.5,2.5];
ped_Width = [0,1];

% Both the Pedestrian and vehicle detector is then configured with the
sensor and the witdth value set
Ped_detector = configureDetectorMonoCamera(Ped_detector, sensor,
ped_Width);
Vehicle_detector = configureDetectorMonoCamera(Vehicle_detector,
sensor, vehicle_Width);

 % A Multi-Object tracker which includes the Kalman filter is then
initialized.
[pedes_tracker, Pedes_position_Selector] = pedestrian_setup_tracker();
[vehicle_tracker, vehicle_position_Selector] = vehicle_setup_tracker();

% The Video file to be run is then loaded
video_File   = 'car_wall.mp4';
detect_video_Reader = VideoReader(video_File);
videoPlayer = vision.DeployableVideoPlayer();

current_time_Step = 0;
snapshot = [];
snapTimeStamp = 120;
% The loop continues till the video reader has frames available
cont = hasFrame(detect_video_Reader);
while cont
   % The frame counters are updated
```

```
    current_time_Step = current_time_Step + 1;

    % Read a frame
    detect_frame = readFrame(detect_video_Reader);

    % Implement both the detectors and the returned results into an
object
    % which is required by the multiObjectTracker function
    pedest_detections = pedestrian_detect_objects(Ped_detector,
detect_frame, current_time_Step);
    vehicle_detections = vehicle_detect_objects(Vehicle_detector,
detect_frame, current_time_Step);


    % The tracks are returned after being updated  for the
'currentStep' time.

    if current_time_Step == 1
        costMatrix1 = zeros(0, numel(pedest_detections));
        [pedestrian_confirmed_Tracks,~,allTracks] =
updateTracks(pedes_tracker,pedest_detections,
        current_time_Step, costMatrix1);
    else
        costMatrix1 = ped_detection_Cost(allTracks, pedest_detections,
...
            Pedes_position_Selector,
pedes_tracker.AssignmentThreshold);
        [pedestrian_confirmed_Tracks,~,allTracks] =
updateTracks(pedes_tracker, pedest_detections, current_time_Step, ...
            costMatrix1);
    end
    vehicle_confirmed_Tracks = updateTracks(vehicle_tracker,
vehicle_detections, current_time_Step);


    % The vehicles and pedestrians which are far away are then removed
away.
    pedestrian_confirmed_Tracks =
pedestrians_remove_noisy_tracks(pedestrian_confirmed_Tracks,
Pedes_position_Selector, sensor.Intrinsics.ImageSize);
    vehicle_confirmed_Tracks =
vehicle_remove_noisy_tracks(vehicle_confirmed_Tracks,
vehicle_position_Selector, sensor.Intrinsics.ImageSize);


    % The annotations are added to the frame for it to be visualized
    frame_With_pedestrians = pedestrian_insert_box(detect_frame,
pedestrian_confirmed_Tracks, Pedes_position_Selector, sensor);
    % The frame with the pedestrian annotations is then fed to the
    % vehicle annotation box inserter
    final_frame = vehicle_insert_box(frame_With_pedestrians,
vehicle_confirmed_Tracks, vehicle_position_Selector, sensor);

    % Display the annotated frame.
    videoPlayer(final_frame);

    % Exit the loop if the video player figure is closed by user.
    cont = hasFrame(detect_video_Reader) && isOpen(videoPlayer);
end
```

```
% --------------------------------FUNCTIONS-------------------------------
---------

% --------------------------Vehicle Detection ------------------------
---------



% ----------------------vehicle_setup_tracker()-----------------------
------
% Vehicle_setup_tracker()- creates a multiObjectTracker
% to track many vehicles using the Kalman filters
function [vehicle_tracker, vehicle_position_selector] =
vehicle_setup_tracker()
    % The Tracker object is created
    vehicle_tracker = multiObjectTracker('FilterInitializationFcn',
@vehicle_init_Bbox_Filter, ...
        'AssignmentThreshold', 50, ...
        'NumCoastingUpdates', 5, ...
        'ConfirmationParameters', [3 5]);

    % The State vector is: [x; vx; y; vy; w; vw; h; vh]
    % [x;y;w;h] = positionSelector * State
    vehicle_position_selector = [1 0 0 0 0 0 0 0; ...
                         0 0 1 0 0 0 0 0; ...
                         0 0 0 0 1 0 0 0; ...
                         0 0 0 0 0 0 1 0];
end




% ----------------------vehicle_init_Bbox_filter----------------------
------
% The Vehicle_init_Bbox_Filter defines a Kalman filter to help filter
bounding box measurement
function vehicle_filter = vehicle_init_Bbox_Filter(vehicle_detection)
% First the model and state is defined
% A constant velocity model is used for making the bounding box
%   The state is [x; vx; y; vy; w; wv; h; hv]
%   The state transition matrix is:
%       [1 dt 0  0 0  0 0  0;
%        0  1 0  0 0  0 0  0;
%        0  0 1 dt 0  0 0  0;
%        0  0 0  1 0  0 0  0;
%        0  0 0  0 1 dt 0  0;
%        0  0 0  0 0  1 0  0;
%        0  0 0  0 0  0 1 dt;
%        0  0 0  0 0  0 0  1]
%   dt = 1 is Assumed. In this case time-variant transition is  not
%   considered
    deltat = 1;
    cvel =[1 deltat; 0 1];
    A_car = blkdiag(cvel, cvel, cvel, cvel);
```

```
%  Then noise is defined, this is the part that the model
%  does not take into account. In a non-time variant model which is
used the
%  Acceleration is neglected
    G1d = [deltat^2/2; deltat];
    Q1d = G1d*G1d';
    Q = blkdiag(Q1d, Q1d, Q1d, Q1d);

%  The measurement model is defined
%  Only the position ([x;y;w;h]) is measured.
%  The measurement model is
    H_model_car = [1 0 0 0 0 0 0 0; ...
         0 0 1 0 0 0 0 0; ...
         0 0 0 0 1 0 0 0; ...
         0 0 0 0 0 0 1 0];

%  Then the sensor measurements is mapped to an initial state vector.
%  The velocity components are initialized to 0
    vehicle_state = [vehicle_detection.Measurement(1); ...
              0; ...
              vehicle_detection.Measurement(2); ...
              0; ...
              vehicle_detection.Measurement(3); ...
              0; ...
              vehicle_detection.Measurement(4); ...
              0];

% The sensor measurment is mapped to a state covariance matrix
%   For the parts of the state that the sensor measured directly, use
the
%   corresponding measurement noise components. For the parts that the
%   sensor does not measure, assume a large initial state covariance.
%   That way, future detections can be assigned to the track.
    vec_L = 100; % Large value
    vehicle_state_cov = diag([vehicle_detection.MeasurementNoise(1,1),
...
                      vec_L, ...
                      vehicle_detection.MeasurementNoise(2,2), ...
                      vec_L, ...
                      vehicle_detection.MeasurementNoise(3,3), ...
                      vec_L, ...
                      vehicle_detection.MeasurementNoise(4,4), ...
                      vec_L]);

% Finally a filter is created, for this model a linear
% Filter is used

    vehicle_filter = trackingKF(...
        'StateTransitionModel', A_car, ...
        'MeasurementModel', H_model_car, ...
        'State', vehicle_state, ...
        'StateCovariance', vehicle_state_cov, ...
        'MeasurementNoise', vehicle_detection.MeasurementNoise, ...
        'ProcessNoise', Q);
End
```

```
 % ----------------------vehicle_detect_objects()----------------------
------


% The vehicle_detect_objects- detects vehicles in an image.
function vehicle_detections = vehicle_detect_objects(car_detector,
car_frame, frameCount)
    % The detector is run on an image and
    % a list of bounding boxes is returned: [x, y, w, h]
    vech_bboxes = detect(car_detector, car_frame);

    % The measurement noise is defined
    vech_L = 100;
    vech_measurementNoise = [vech_L 0  0  0; ...
                        0 vech_L  0  0; ...
                        0 0 vech_L/2 0; ...
                        0 0  0 vech_L/2];

    % The detections is then formulated as a list of objectDetection
reports.
    vehicle_num_detections = size(vech_bboxes, 1);
    vehicle_detections = cell(vehicle_num_detections, 1);
    for i = 1:vehicle_num_detections
        vehicle_detections{i} = objectDetection(frameCount,
vech_bboxes(i, :), ...
            'MeasurementNoise', vech_measurementNoise);
    end
end




% -----------------vehicle_remove_noisy_tracks-------------------------
--

%  vehicle_remove_noisy_tracks function removes noisy tracks.
%  A track might be noisy if its predicted bounding box is quite small
%  The small bounding box implies that the car is quite far
function vechicle_tracks = vehicle_remove_noisy_tracks(vechicle_tracks,
car_position_selector, imageSize)

    if isempty(vechicle_tracks)
        return
    end

    % The position is extracted from the tracks
    vehicle_positions = getTrackPositions(vechicle_tracks,
car_position_selector);
    % The track is not valid if the bounding box is too small.
    invalid = ( vehicle_positions(:, 1) < 1 | ...
                vehicle_positions(:, 1) + vehicle_positions(:, 3) >
imageSize(2) | ...
                vehicle_positions(:, 3) <= 20 | ...
                vehicle_positions(:, 4) <= 20 );
    vechicle_tracks(invalid) = [];
end
```

```
% ----------------------vehicle_insert_box()--------------------------
-------


% The vehicle_insert_box funtion adds bounding boxes in the image and
displays
% the track's position in world coordinates.
function vechicle_I = vehicle_insert_box(vechicle_I, tracks,
positionSelector, sensor)

    if isempty(tracks)
        return
    end

    labels = cell(numel(tracks), 1);

    % Retrieve positions of bounding boxes.
    bboxes = getTrackPositions(tracks, positionSelector);


    for i = 1:numel(tracks)
        box = bboxes(i, :);

        % Convert to vehicle coordinates using the sensor mono camera
object.
        xyVehicle = imageToVehicle(sensor, [box(1)+box(3)/2,
box(2)+box(4)]);

        % These are the labels which will be displayed on the image
        labels{i} = sprintf('X=%.1f,
Y=%.1f',xyVehicle(1),xyVehicle(2));

    end

    % The Value of alpha for the vehicle is defined here, It is the
minimum
    % safe distance after which the vehicle will send message to the
ROS
    % controller to apply brakes
    alpha_vehicle = 2.0 ;

    % The minimum distance of the vehicle is used and is compared
agaisnt
    % the alpha value
    if sqrt(xyVehicle(1)*xyVehicle(1)+xyVehicle(2)*xyVehicle(2)) <
alpha_vehicle
        vechicle_I = insertObjectAnnotation(vechicle_I, 'rectangle',
bboxes, labels, 'Color', 'red', ...
        'FontSize', 15, 'TextBoxOpacity', .8, 'LineWidth', 4);
        % Publish Message to ROS here!

    elseif sqrt(xyVehicle(1)*xyVehicle(1)+xyVehicle(2)*xyVehicle(2)) >
alpha_vehicle
        vechicle_I = insertObjectAnnotation(vechicle_I, 'rectangle',
bboxes, labels, 'Color', 'green', ...
        'FontSize', 15, 'TextBoxOpacity', .8, 'LineWidth', 4);
        % Continue current values
```

```
    end

end



% ----------------------Pedestrian Tracker----------------------------
------



% --------------------pededtrain_setup_tracker() --------------------
-------------


% The Pedestrian_setup_tracker function creates a multiObjectTracker to
track multiple pedestrians with  with Kalman filters.
function [pedestrian_tracker, pedestrian_position_selector] =
pedestrian_setup_tracker()
    % The Tracker object is created
    pedestrian_tracker = multiObjectTracker('FilterInitializationFcn',
@pedestrian_init_Bbox_Filter, ...
        'AssignmentThreshold', 0.999, ...
        'NumCoastingUpdates', 5, ...
        'ConfirmationParameters', [3 5], ...
        'HasCostMatrixInput', true);

    % The State vector is: [x; vx; y; vy; w; vw; h; vh]
    % [x;y;w;h] = positionSelector * State
    pedestrian_position_selector = [1 0 0 0 0 0 0 0; ...
                        0 0 1 0 0 0 0 0; ...
                        0 0 0 0 1 0 0 0; ...
                        0 0 0 0 0 0 1 0];
end









% --------------------pedestrian_init_Bbox_Filter()-------------------
---

% The Pedestrian_init_Bbox_Filter defines a Kalman filter to filter
bounding box measurement.
function pedestrian_filter =
pedestrian_init_Bbox_Filter(pedes_detection)
% First the model and state is defined
% A constant velocity model is used for making the bounding box
%   The state is [x; vx; y; vy; w; wv; h; hv]
```

```
%   The state transition matrix is:
%       [1 dt 0  0 0  0 0  0;
%        0  1 0  0 0  0 0  0;
%        0  0 1 dt 0  0 0  0;
%        0  0 0  1 0  0 0  0;
%        0  0 0  0 1 dt 0  0;
%        0  0 0  0 0  1 0  0;
%        0  0 0  0 0  0 1 dt;
%        0  0 0  0 0  0 0  1]
%   dt = 1 is Assumed. In this case time-variant transition is  not
%   considered
    delta_t = 1;
    cvel =[1 delta_t; 0 1];
    A_Ped = blkdiag(cvel, cvel, cvel, cvel);

%  Then noise is defined, this is the part that the model
%  does not consider. In a non-time, variant model which is used
%  Acceleration is neglected
    G1d = [delta_t^2/2; delta_t];
    Q1d = G1d*G1d';
    Q = blkdiag(Q1d, Q1d, Q1d, Q1d);

%  The measurement model is defined
%  Only the position ([x;y;w;h]) is measured.
%  The measurement model is
    H_model_ped = [1 0 0 0 0 0 0 0; ...
         0 0 1 0 0 0 0 0; ...
         0 0 0 0 1 0 0 0; ...
         0 0 0 0 0 0 1 0];

%  Then the sensor measurements is mapped to an initial state vector.
%  The velocity components are initialized to 0
    state_ped = [pedes_detection.Measurement(1); ...
             0; ...
             pedes_detection.Measurement(2); ...
             0; ...
             pedes_detection.Measurement(3); ...
             0; ...
             pedes_detection.Measurement(4); ...
             0];

% The sensor measurment is mapped to a state covariance matrix For the
parts of the state that  the sensor measured directly, use the
corresponding measurement noise components. For the parts that the
sensor does not measure, assume a large initial state covariance. That
way, future detections can be assigned to the track.
    L_ped = 120; % Large value
    pedestrian_state_Cov = diag([pedes_detection.MeasurementNoise(1,1),
...
                     L_ped, ...
                     pedes_detection.MeasurementNoise(2,2), ...
                     L_ped, ...
                     pedes_detection.MeasurementNoise(3,3), ...
                     L_ped, ...
                     pedes_detection.MeasurementNoise(4,4), ...
                     L_ped]);
 % Finally, a filter is created, for this model a linear Filter is used
    pedestrian_filter = trackingKF(...
        'StateTransitionModel', A_Ped, ...
```

```
        'MeasurementModel', H_model_ped, ...
        'State', state_ped, ...
        'StateCovariance', pedestrian_state_Cov, ...
        'MeasurementNoise', pedes_detection.MeasurementNoise, ...
        'ProcessNoise', Q);
end



% -----------------pedestrian_detect_obstacles-------------------------
-

% The function pedestrain_detect_objects then detects people in an
image.
function pedestrian_detections =
pedestrian_detect_objects(ped_detector, ped_frame, frameCount)
    % The detector is run on an image and
    % a list of bounding boxes is returned: [x, y, w, h]
    ped_bboxes = detect(ped_detector, ped_frame);

    % The measurement noise is defined
    ped_L = 100;
    measurementNoise = [ped_L 0   0   0; ...
                        0 ped_L   0   0; ...
                        0 0 ped_L/2 0; ...
                        0 0   0 ped_L/2];

    % The detections is then formulated as a list of objectDetection
reports.
    ped_num_detections = size(ped_bboxes, 1);
    pedestrian_detections = cell(ped_num_detections, 1);
    for i = 1:ped_num_detections
        pedestrian_detections{i} = objectDetection(frameCount,
ped_bboxes(i, :), ...
            'MeasurementNoise', measurementNoise);
    end
end




% -----------------pedestrians_remove_noisy_tracks()------------------
---


%  pedestrians_remove_noisy_tracks function removes noisy tracks.
%  A track might be noisy if its predicted bounding box is quite small
%  The small bounding box implies that the person is quite far
function pedestrian_tracks =
pedestrians_remove_noisy_tracks(pedestrian_tracks,
ped_position_selector, imageSize)

    if isempty(pedestrian_tracks)
        return
    end

    % The position is extracted from the tracks.
```

```
    ped_positions = getTrackPositions(pedestrian_tracks,
ped_position_selector);
    % The track is not valid if the bounding box is too small
    ped_invalid_track = ( ped_positions(:, 1) < 1 | ...
                ped_positions(:, 1) + ped_positions(:, 3) >
imageSize(2) | ...
                ped_positions(:, 3) <= 5 | ...
                ped_positions(:, 4) <= 10 );
    pedestrian_tracks(ped_invalid_track) = [];
end




% ----------------------pedestrian_insert_box()---------------------

% The pedestrain_insert_box funtion adds bounding boxes in the image
and displays
% the track's position in world coordinates.
function pedestrain_I = pedestrian_insert_box(pedestrain_I,
pedes_tracks, pedes_position_selector, sensor)

    if isempty(pedes_tracks)
        return
    end

    labels = cell(numel(pedes_tracks), 1);
    % Retrieve positions of bounding boxes.
    bboxes = getTrackPositions(pedes_tracks, pedes_position_selector);

    for i = 1:numel(pedes_tracks)
        box = bboxes(i, :);

        xyImageLoc = [box(1)+box(3)/2, box(2)+box(4)];
        % Constrain the image point to be within the image boarder.
        xyImageLoc(1) = min(max(xyImageLoc(1), 1), size(pedestrain_I,
2));
        xyImageLoc(2) = min(xyImageLoc(2), size(pedestrain_I, 1));

        % Convert to vehicle coordinates using the sensor mono camera
object.
        xyVehicle = imageToVehicle(sensor, xyImageLoc);

        % These are the labels which will be displayed on the image
        labels{i} = sprintf('x=%.1f,y=%.1f',xyVehicle(1),xyVehicle(2));
    end


    % The Value of alpha for the pedestrian is defined here, It is the
minimum
    % safe distance after which the vehicle will send message to the
ROS
    % controller to apply brakes
   alpha_pedestrian = 2.0 ;
   if sqrt(xyVehicle(1)*xyVehicle(1)+xyVehicle(2)*xyVehicle(2)) <
alpha_pedestrian
        pedestrain_I = insertObjectAnnotation(pedestrain_I,
'rectangle', bboxes, labels, 'Color', 'red', ...
```

```
        'FontSize', 15, 'TextBoxOpacity', .8, 'LineWidth', 4);
        %disp("Pedestrian very close!! Collision possible!")
        % Send message to ROS
    elseif sqrt(xyVehicle(1)*xyVehicle(1)+xyVehicle(2)*xyVehicle(2)) >
alpha_pedestrian
        pedestrain_I = insertObjectAnnotation(pedestrain_I,
'rectangle', bboxes, labels, 'Color', 'green', ...
        'FontSize', 15, 'TextBoxOpacity', .8, 'LineWidth', 4);
        %disp("All Safe!")
    end
end




% --------------------pedestrian_detection_cost()----------------------

% The ped_detection_Cost function detectionToTrackCost computes the
% customized cost for detections to track assignment.
function ped_cost_matrix = ped_detection_Cost(tracks, detections, ...
                                positionSelector, threshold)

    if isempty(tracks) || isempty(detections)
        ped_cost_matrix = zeros(length(tracks), length(detections));
        return
    end

    % The overlap ratio is calculated between the predicted boxes
    % and the detected boxes, and a cost is assigned to each detection
    % for each track. The cost is the least when the predicted box is
    % totally aligned with the detected box

    % Retrieve the positions of the bounding boxes.
    ped_track_bboxes = getTrackPositions(tracks, positionSelector);
    % The width and height are checked for a positive value before
    % they are used for computing the box overlap ratio.
    ped_track_bboxes(:, 3) = max(eps, ped_track_bboxes(:, 3));
    ped_track_bboxes(:, 4) = max(eps, ped_track_bboxes(:, 4));

    % Extract the detected bounding box from all the detections.
    ped_alldetections = [detections{:}];
    bboxes = reshape([ped_alldetections(:).Measurement], 4,
length(detections))';

    % Compute all assignment costs.
    ped_cost_matrix = 1 - bboxOverlapRatio(ped_track_bboxes, bboxes);
    % If there is very little overlap the cost is set to infinite
    ped_cost_matrix(ped_cost_matrix(:) > threshold) = Inf;
end
```

Appendix D
CODE FOR PARKING VALET SIMULATION

```
%------------------Initialize Code------------------------
% First a map with already predefined obstacles and
% walls is loaded. This map is an inbuilt map of a parking lot
% There are 3 layers in the map, the vehicles, the boundary and the
% Parking obstacles. The White part of the map is free space
% The black part are onstacles
map_Layers = loadParkingLotMapLayers;
plotMapLayers(map_Layers)

% For ease the 3 layers are then combined
cost_Vehicle_map = combineMapLayers(map_Layers);

figure
plot(cost_Vehicle_map)
legend off



%%
% The area covered by the map is 75 meters in width and 50 meters in
% Length.

% The dimentions of the vehicle are set here along with the maximum
steering angle
% The dimentions used here are of the E20 vehicle
vehicle_dims      = vehicleDimensions(3.28,1.54,1.56,'WheelBase',1.9);
max_Steering_Angle = 35; % in degrees


% The Initial pose of the vehicle is defined in the world coordiantes

vehicle_current_Pose = [4 12 0]; % [x, y, theta]


% The File with the static route for the vehicle is then loaded
% The table also is in accordance to the 3 variables, Start pose, End
Pose
% and Angle( theta)
datapath = load('routePlan.mat');
route_Plan = data.routePlan

%%
% This plots the car at each of the pose described in the route plan

hold on
```

```
helperPlotVehicle(vehicle_current_Pose, vehicle_dims, 'DisplayName',
'Current Pose')
legend

for n = 1 : 4
    % Extract the waypoint from the route table
    vehicle_Pose = route_Plan{n, 'EndPose'};

    % Plot the vehicle at the point
    legendPlot = sprintf('Goal %i', n);
    helperPlotVehicle(vehicle_Pose, vehicle_dims, 'DisplayName',
legendPlot);
end
hold off
% A behavioral planner helper object is created
% The Path planner requests a stream of navigation tasks from this
planner until
% the destination is reached.
behavioralPlanner = HelperBehavioralPlanner(cost_Vehicle_map,
route_Plan, ...
    vehicle_dims, max_Steering_Angle);


%%-----------------------------Motion Planning-------------------------
--

% This creates an object to make a path planner using the RRT*
algorithm,
% to find a Collison free path for the vehicle. The poses generated are
% made sure that they are kinematically feasible.
motion_Vehicle_Planner = pathPlannerRRT(cost_Vehicle_map,
'MinIterations', 1000, ...
    'ConnectionDistance', 10, 'MinTurningRadius', 20);

%%
% Executes a plan from the current pose to the first goal point
goal_Vehicle_Pose = route_Plan{1, 'EndPose'};
referance_Path = plan(motion_Vehicle_Planner, vehicle_current_Pose,
goal_Vehicle_Pose);

%%
% The reference path consists of a sequence of path segments. Each path
% then helps to connect to the other segment
referance_Path.PathSegments



% Retrieve transition poses and directions from the planned path.
[transition_Vehicle_Poses, directions_planned_path] =
interpolate(referance_Path);

% Visualize the planned path
plot(motion_Vehicle_Planner)
```

```
%%-------------Path Smoothing and Trajectory Creation-----------------


% The path between two goal points is then fitted with a cubic spline
polynomial. This is done to make the path smooth and differentiable


% Create an object to use spline fitting on the path
spline_vehicle_Fitter = HelperCubicSplineFit(transition_Vehicle_Poses,
directions_planned_path);

% Number of spline points
spline_vehicle_Fitter.NumPoints = 1000;

% Fit the spline
[ref_spline_Poses, directions_planned_path, ref_1_Path_splineLengths] =
spline_vehicle_Fitter();

% Plot the smoothed path
hold on
hSmoothPath = plot(ref_spline_Poses(:, 1), ref_spline_Poses(:, 2), 'r',
'LineWidth', 2, ...
    'DisplayName', 'Smoothed Path');
hold off


% Specify staring, maximum, and terminal speeds so that the vehicle
starts
% stationary.
max_vehicle_Speed   = 5; % in meters/second
Vehicle_start_Speed = 0; % in meters/second
Vehicle_end_Speed   = 0; % in meters/second

%%
% Create an object to generate the speed profile based the smoothed
path.
speed_smooth_Profile_Generator =
HelperSpeedProfileGenerator(Vehicle_start_Speed, Vehicle_end_Speed,
max_vehicle_Speed);


% Generate a speed profile based on the accumulated lengths along the
% path and the previously specified speeds.
ref_Vehicle_Speed_profile =
speed_smooth_Profile_Generator(ref_1_Path_splineLengths);

% Plot the Speed Profile
plotSpeedProfile(ref_1_Path_splineLengths, ref_Vehicle_Speed_profile,
max_vehicle_Speed)
```

```
%% -----------------Vehicle Control and Simulation--------------------

% Create the E20/ vehicle simulation
E2o_Simulator = HelperVehicleSimulator(cost_Vehicle_map, vehicle_dims);

% Set the vehicle pose and velcity to the defined values
E2o_Simulator.setVehiclePose(vehicle_current_Pose);
Vehicle_current_Vel = 0;
E2o_Simulator.setVehicleVelocity(Vehicle_current_Vel);

% Configure the simulation to show the trajectory
E2o_Simulator.showTrajectory(true);

%  Hide the figure
hideFigure(E2o_Simulator);

% Create an object find the reference pose, reference velocity
% and driving direction for the controller.

path_Ref_Analyzer = HelperPathAnalyzer(ref_spline_Poses,
ref_Vehicle_Speed_profile, directions_planned_path, ...
    'Wheelbase', vehicle_dims.Wheelbase);


% Generate an object to control the velocity of the vehicle and specify
the sample time.
sampleTime = 0.05;
lonController = HelperLongitudinalController('SampleTime', sampleTime);

%%
% Use the object to ensure fixed-rate execution of the feedback
% controller. Use a control rate to be consistent with the longitudinal
% controller.
controlRate = HelperFixedRate(1/sampleTime); % in Hertz
```

```
% -------Make a while loop till the goal position is not reached-------

if_reach_Goal = false;

while ~if_reach_Goal
    % Calculate the reference pose and the corresponding velocity on
the
    % path
    [referance_Pose_vehicle, referance_Vel_vehicle, direction_car] =
path_Ref_Analyzer(vehicle_current_Pose, Vehicle_current_Vel);

    % Update driving direction for the vehicle
    updateDrivingDirection(E2o_Simulator, direction_car);

    % Calculate the steering command
    steering_Angle_Vehicle =
lateralControllerStanley(referance_Pose_vehicle, vehicle_current_Pose,
Vehicle_current_Vel, ...
        'Direction', direction_car, 'Wheelbase',
vehicle_dims.Wheelbase);

    % Compute acceleration/deceleration
    lonController.Direction = direction_car;
    [accelCmd, decelCmd] = lonController(referance_Vel_vehicle,
Vehicle_current_Vel);
    drive(E2o_Simulator, accelCmd, decelCmd, steering_Angle_Vehicle);

    % Conform if the car reaches the goal
    if_reach_Goal = helperGoalChecker(goal_Vehicle_Pose,
vehicle_current_Pose, Vehicle_current_Vel, Vehicle_end_Speed,
direction_car);

    % Wait for fixed-rate execution
    waitfor(controlRate);

    % Get current pose and velocity of the vehicle
    vehicle_current_Pose  = getVehiclePose(E2o_Simulator);
    Vehicle_current_Vel   = getVehicleVelocity(E2o_Simulator);
end

% Show vehicle simulation figure
showFigure(E2o_Simulator);


%%------------------------Execute a Complete Plan-------------------

% Set the vehicle pose back to the initial starting point
vehicle_current_Pose = [4 12 0];
E2o_Simulator.setVehiclePose(vehicle_current_Pose);

% Set initial Velocity to 0
Vehicle_current_Vel  = 0;
E2o_Simulator.setVehicleVelocity(Vehicle_current_Vel);
```

```
while ~reachedDestination(behavioralPlanner)

    % Request next maneuver from behavioral layer
    [vehicle_next_Goal, goal_planner_Config, goal_speed_Config] =
requestManeuver(behavioralPlanner, ...
        vehicle_current_Pose, Vehicle_current_Vel);

    % Configure the motion planner
    configurePlanner(motion_Vehicle_Planner, goal_planner_Config);

    % Plan a path using RRT* planner to the next goal pose
    referance_Path = plan(motion_Vehicle_Planner, vehicle_current_Pose,
vehicle_next_Goal);

    % Check if the path made is collision free.
    is_Replan_Needed = ~checkPathValidity(referance_Path,
cost_Vehicle_map);
    if is_Replan_Needed
        warning('Unable to find a Plan! Please re plan.')

        % Request behavioral planner to re-plan
        replanNeeded(behavioralPlanner);
        continue;
    end

    % Retrieve transition poses and directions from the planned path
    [transition_Vehicle_Poses, directions_planned_path] =
interpolate(referance_Path);

    % Assign poses and directions to the spline fitting object
    spline_vehicle_Fitter.Poses      = transition_Vehicle_Poses;
    spline_vehicle_Fitter.Directions = directions_planned_path;

    % Fit the spline
    [ref_spline_Poses, directions_planned_path,
ref_1_Path_splineLengths] = spline_vehicle_Fitter();

    % Generate a trajectory
    configureSpeedProfileGenerator(speed_smooth_Profile_Generator,
goal_speed_Config);
    ref_Vehicle_Speed_profile =
speed_smooth_Profile_Generator(ref_1_Path_splineLengths);

    % Initialize the path analyzer
    path_Ref_Analyzer.RefPoses     = ref_spline_Poses;
    path_Ref_Analyzer.Directions   = directions_planned_path;
    path_Ref_Analyzer.SpeedProfile = ref_Vehicle_Speed_profile;

    % Reset the longitudinal controller
    reset(lonController);

    if_reach_Goal = false;
```

```
    % till it does not reach the goal
    while ~if_reach_Goal
        % Find the reference pose using the velocity
        [referance_Pose_vehicle, referance_Vel_vehicle, direction_car]
= path_Ref_Analyzer(vehicle_current_Pose, Vehicle_current_Vel);

        % Update driving direction
        updateDrivingDirection(E2o_Simulator, direction_car);

        % Compute steering command
        steering_Angle_Vehicle =
lateralControllerStanley(referance_Pose_vehicle, vehicle_current_Pose,
Vehicle_current_Vel, ...
            'Direction', direction_car, 'Wheelbase',
vehicle_dims.Wheelbase);

        % Compute acceleration/deacceleration
        lonController.Direction = direction_car;
        [accelCmd, decelCmd] = lonController(referance_Vel_vehicle,
Vehicle_current_Vel);

        % Simulate the vehicle using the controller outputs
        drive(E2o_Simulator, accelCmd, decelCmd,
steering_Angle_Vehicle);

        % Check if the vehicle reaches the goal
        if_reach_Goal = helperGoalChecker(vehicle_next_Goal,
vehicle_current_Pose, Vehicle_current_Vel, goal_speed_Config.EndSpeed,
direction_car);

        % Wait for fixed-rate execution
        waitfor(controlRate);

        % Get current pose and velocity of the vehicle
        vehicle_current_Pose  = getVehiclePose(E2o_Simulator);
        Vehicle_current_Vel   = getVehicleVelocity(E2o_Simulator);
    end
end

% Show vehicle simulation figure
showFigure(E2o_Simulator);
```

```
% -------------------------Helper Functions--------------------------


% -------------------------LoadparkingLotMapLayers()-------------------

% This function loads the 3 images using which the 3 layer
% occupancy map is made
function Load_map_Layers = loadParkingLotMapLayers()

Load_map_Layers.StationaryObstacles = imread('stationary.bmp');
Load_map_Layers.RoadMarkings        = imread('road_markings.bmp');
Load_map_Layers.ParkedCars          = imread('parked_cars.bmp');
end



% -------------------------plotmaplayers()--------------------------

%Plot the Map
function plotMapLayers(mapLayers)
figure
cellOfMaps = cellfun(@imcomplement, struct2cell(mapLayers),
'UniformOutput', false);
montage( cellOfMaps, 'Size', [1 numel(cellOfMaps)], 'Border', [5 5],
'ThumbnailSize', [300 NaN] )
title('Map Layers - Stationary Obstacles, Road markings, and Parked
Cars')
end




 % -------------------------combineMapLayers()-------------------------
-

% Overlap the 3 images to combine to form 1 image
function costmap_world = combineMapLayers(mapLayers)

world_combined_Map = mapLayers.StationaryObstacles +
mapLayers.RoadMarkings + ...
    mapLayers.ParkedCars;
world_combined_Map = im2single(world_combined_Map);

res = 0.5; % meters
costmap_world = vehicleCostmap(world_combined_Map, 'CellSize', res);
end
```

```
% -------------------------ConfigurePanner()-------------------------

% Configure the path planner
function configurePlanner(pathPlanner, config)

fieldNames = fields(config);
for n = 1 : numel(fieldNames)
    pathPlanner.(fieldNames{n}) = config.(fieldNames{n});
end
end



% -------------------------configureSpeedProfileGenerator()-----------
--------------


% Configure speed profile generator
function configureSpeedProfileGenerator(speedProfiler, config)
fieldNames = fields(config);
for n = 1 : numel(fieldNames)
    speedProfiler.(fieldNames{n}) = config.(fieldNames{n});
end
end


% -------------------------plotSpeedProfile()-------------------------

% Plot speed profile
function plotSpeedProfile(refPathLengths, refSpeeds, maxSpeed)

% Plot reference speeds along length of the path
plot(refPathLengths, refSpeeds, 'LineWidth', 2);

% Plot a line to display maximum speed
hold on
line([0;refPathLengths(end)], [maxSpeed;maxSpeed], 'Color', 'r')
hold off

% Set axes limits
buffer = 2;
xlim([0 refPathLengths(end)]);
ylim([0 maxSpeed + buffer])

% Add labels
xlabel('Path Length (m)');
ylabel('Speed (m/s)');

% Add legend and title
legend('Speed Profile', 'Max Speed')
title('Generated speed profile')
end
```

Appendix E

CODE FOR CNN BASED PARKING SIGN DETECTION

```
# Import all the libraries used
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
import cv2
from sklearn.model_selection import train_test_split
import pickle
import os
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D



################# Parameters #####################

path = "myData" # folder with all the class folders,( 43 class folders each with
a
                # different road sign dataset of image.
labelFile = 'labels.csv' # file with all names of classes
batch_size = 50  # Batch size set
steps_per_epoch = 2000
epochs = 10
imageDimesions = (32,32,3)
testRatio = 0.2    # 20% of total images go to test
validationRatio = 0.2 # 20% of the remaining go to validation set
##################################################


# Importing images from the folder
count = 0
images = []
```

```
    classNo = []
    myList = os.listdir(path)
    print("Total Number of Classes Detected:",len(myList))
    noOfClasses=len(myList)
    print("Importing the class .")
    for x in range (0,len(myList)):
        myPicList = os.listdir(path+"/"+str(count))
        for y in myPicList:
            curImg = cv2.imread(path+"/"+str(count)+"/"+y)
            images.append(curImg)
            classNo.append(count)
        print(count, end =" ")
        count +=1
    print(" ")
    images = np.array(images)
    classNo = np.array(classNo)

    # Split the data into training, test and validation set
    # X_train the st of images to be trained
    # y_train the corresponding class names
    X_train, X_test, y_train, y_test = train_test_split(images, classNo, test_size=t
estRatio)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train
, test_size=validationRatio)


    # read the CSV file to check the labels
    data=pd.read_csv(labelFile)
    print("data shape ",data.shape,type(data))



    # Preprocessing

    #converting the RGB images to grayscale, to remove the dependancy of colors
    def grayscale(img):
        img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        return img
    # Equalizing the brightness and lighting of he images
    def equalize(img):
        img =cv2.equalizeHist(img)
        return img
    def preprocessing(img):
        img = grayscale(img)
        img = equalize(img)
        #Normalize the pixel values between 0 and 1
        img = img/255
```

```
        return img


    #Apply preprocessing to all the datasets
    X_train=np.array(list(map(preprocessing,X_train)))
    X_validation=np.array(list(map(preprocessing,X_validation)))
    X_test=np.array(list(map(preprocessing,X_test)))


    #As converting to a gryscale removes the last channel and reduces the diemntion
a extra dimention of 1 is added
    X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
    X_validation=X_validation.reshape(X_validation.shape[0],X_validation.shape[1],X_
validation.shape[2],1)
    X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)


    #Augenting images to make them more random
    dataGen= ImageDataGenerator(width_shift_range = 0.2,
                                height_shift_range = 0.2,
                                zoom_range = 0.1,
                                shear_range = 0.3,
                                rotation_range = 20)
    #Fitting the training data and creating batched
    dataGen.fit(X_train)
    batches= dataGen.flow(X_train,y_train, batch_size = 50)
    X_batch,y_batch = next(batches)

    # To convert the Y labels into one hot vectors
    y_train = to_categorical(y_train,noOfClasses)
    y_validation = to_categorical(y_validation,noOfClasses)
    y_test = to_categorical(y_test,noOfClasses)

    # Define the CNN model
    def myModel():
        no_Of_Filters=60
        # The size of the kernel defined which will move to get the features.
        size_of_Filter=(5,5)

        size_of_Filter2=(3,3)
        # Scale down the feature map, this is done to reduce overfitting
        size_of_pool=(2,2)
        #Hidden layers nodes
        no_Of_Nodes = 500
        model= Sequential()
        model.add((Conv2D(60,(5,5),input_shape=(imageDimesions[0],imageDimesions[1],
1),activation='relu')))
```

```
        model.add((Conv2D(60,(5,5), activation='relu')))

        model.add(MaxPooling2D(pool_size =(2,2)))

        model.add((Conv2D(30, (3,3),activation='relu')))

        model.add((Conv2D(30, (3,3), activation='relu')))

        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Dropout(0.5))

        model.add(Flatten())

        model.add(Dense(500,activation='relu'))

        model.add(Dropout(0.5))
        #Output layer
        model.add(Dense(noOfClasses,activation='softmax'))
        # Compile the model, learning rate as 0.001 and use categorial crossentropy
loss
        model.compile(Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accur
acy'])
        return model


   # Train the model
   model = myModel()
   print(model.summary())
   history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size),
steps_per_epoch = steps_per_epoch,epochs = epochs,validation_data=(X_validation,y_
validation),shuffle=1)

   # PLOT results
   plt.figure(1)
   plt.plot(history.history['loss'])
   plt.plot(history.history['val_loss'])
   plt.legend(['training','validation'])
   plt.title('Loss vs Epochs')
   plt.xlabel('No. of Epochs')
   plt.ylabel('Loss Value')

   plt.figure(2)
   plt.plot(history.history['accuracy'])
   plt.plot(history.history['val_accuracy'])
   plt.legend(['training','validation'])
   plt.title('Accuracy vs Epochs')
```

```
plt.xlabel('No. of Epochs')
plt.ylabel('Accuracy Value')
plt.show()
score =model.evaluate(X_test,y_test,verbose=0)

#Pprint Confususion matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
rounded_labels=np.argmax(y_test, axis=1)
rounded_labels[1]
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(rounded_labels, rounded_predictions)

rounded_predictions = model.predict_classes(X_test)

print 'Confusion Matrix :'
print(results)
print 'Accuracy Score :',accuracy_score(actual, predicted)
print 'Report : '
print classification_report(actual, predicted)
results = confusion_matrix(actual, predicted)
a = classification_report(rounded_labels, rounded_predictions)
#Print the classification report
print 'Confusion Matrix :'
print(results)
print 'Accuracy Score :',accuracy_score(actual, predicted)
print 'Report : '
classification_report(y_test, resultz)


#Save the model

pickle_out= open("model.p","wb")
pickle.dump(model,pickle_out)
pickle_out.close()
```

Appendix F

LIVE VIDEO TEST CODE FOR CNN BASED PARKING SIGN DETECTION

```
# Import all the libraries required for this code
import numpy as np
import cv2
import pickle

##############################################
 # Set the camera resolution
frame_Width= 640
frame_Height = 480
brightness = 180
# Set the probability score above which the model shall identify the image
thresh = 0.75
font = cv2.FONT_HERSHEY_SIMPLEX


# Turn on the video
cap = cv2.VideoCapture(0)
cap.set(3, frame_Width)
cap.set(4, frame_Height)
cap.set(10, brightness)
# Impport the trained model
pickle_in=open("model.p","rb")
model=pickle.load(pickle_in)
def grayscale(img):
    img = cv2.cvtColor(img,cv2.COLOR BGR2GRAY)
    return img
def equalize(img):
    img =cv2.equalizeHist(img)
    return img
def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img
def getCalssName(classNo):
    if   classNo == 0: return 'Speed Limit 20 km/h'
    elif classNo == 1: return 'Speed Limit 30 km/h'
```

```
    elif classNo == 2: return 'Speed Limit 50 km/h'
    elif classNo == 3: return 'Speed Limit 60 km/h'
    elif classNo == 4: return 'Speed Limit 70 km/h'
    elif classNo == 5: return 'Speed Limit 80 km/h'
    elif classNo == 6: return 'End of Speed Limit 80 km/h'
    elif classNo == 7: return 'Speed Limit 100 km/h'
    elif classNo == 8: return 'Speed Limit 120 km/h'
    elif classNo == 9: return 'No passing'
    elif classNo == 10: return 'No passing for vechiles over 3.5 metric tons'
    elif classNo == 11: return 'Right-of-way at the next intersection'
    elif classNo == 12: return 'Priority road'
    elif classNo == 13: return 'Yield'
    elif classNo == 14: return 'Stop'
    elif classNo == 15: return 'No vechiles'
    elif classNo == 16: return 'Vechiles over 3.5 metric tons prohibited'
    elif classNo == 17: return 'No entry'
    elif classNo == 18: return 'General caution'
    elif classNo == 19: return 'Dangerous curve to the left'
    elif classNo == 20: return 'Dangerous curve to the right'
    elif classNo == 21: return 'Double curve'
    elif classNo == 22: return 'Bumpy road'
    elif classNo == 23: return 'Slippery road'
    elif classNo == 24: return 'Road narrows on the right'
    elif classNo == 25: return 'Road work'
    elif classNo == 26: return 'Traffic signals'
    elif classNo == 27: return 'Pedestrians'
    elif classNo == 28: return 'Children crossing'
    elif classNo == 29: return 'Bicycles crossing'
    elif classNo == 30: return 'Beware of ice/snow'
    elif classNo == 31: return 'Wild animals crossing'
    elif classNo == 32: return 'End of all speed and passing limits'
    elif classNo == 33: return 'Turn right ahead'
    elif classNo == 34: return 'Turn left ahead'
    elif classNo == 35: return 'Ahead only'
    elif classNo == 36: return 'Go straight or right'
    elif classNo == 37: return 'Go straight or left'
    elif classNo == 38: return 'Keep right'
    elif classNo == 39: return 'Keep left'
    elif classNo == 40: return 'Roundabout mandatory'
    elif classNo == 41: return 'End of no passing'
    elif classNo == 42: return 'End of no passing by vechiles over 3.5metric ton'

while True:
    # Read image from camera
    success, imgOrignal = cap.read()
    # process the frame
    img = np.asarray(imgOrignal)
```

```
    img = cv2.resize(img, (32, 32))
    img = preprocessing(img)
    img = img.reshape(1, 32, 32, 1)
    cv2.putText(imgOrignal, "CLASS: " , (20, 35), font, 0.75, (0, 0, 255), 2, cv2
.LINE_AA)

    # Predict the class
    predictions = model.predict(img)
    classIndex = model.predict_classes(img)
    probabilityValue =np.amax(predictions)
    if probabilityValue > thresh:
          cv2.putText(imgOrignal,str(classIndex)+" "+str(getCalssName(classIndex)
), (120, 35), font,0.75, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.putText(imgOrignal, str(round(probabilityValue*100,2) )+"%", (180, 75
), font, 0.75, (0, 0,255), 2, cv2.LINE_AA)
    cv2.imshow("Result", imgOrignal)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Appendix G

CODE FOR CREATING OCCUPANCY GRID USING MONOCULAR CAMERA

```
%-------Initialize code--------
clc
clear all
close all
%------------Camera callibration-----------------
% intrinsics
% in pixel units
focalLength    = [771.6633 773.6966];
principalPoint = [622.8061 379.3357];
dimensions_image      = [720 1280];

camIntrinsics = cameraIntrinsics(focalLength, principalPoint, dimensions_image);

%extrinsics
height = 1.3;
pitch  = 7;
sensor = monoCamera(camIntrinsics, height, 'Pitch', pitch);

% Downloading the pretrained network
matlab_trained_link =
'https://www.mathworks.com/supportfiles/vision/data/segnetVGG16CamVid.mat';
matlab_trained_directory = fullfile(tempdir,'pretrainedSegNet');
pretrainedSegNet = fullfile(matlab_trained_directory,'segnetVGG16CamVid.mat');

if ~exist(matlab_trained_directory,'dir')
    mkdir(matlab_trained_directory);
    disp('Downloading SegNet');
    websave(pretrainedSegNet,matlab_trained_link);
    disp('Download done.');
end

% Loading the pretrained network
seg_data = load(pretrainedSegNet);
net = seg_data.net;

% The camera view frame
I = imread('road3.jpg');

% Frame segmenting
[C,scores,allScores] = semanticseg(I,net);

% Estimating free space
% Overlay onto the image.
D = labeloverlay(I,C,'IncludedLabels',"Road");

% Show free space on the image.
figure
imshow(D)
```

```
% For each pixel, we get the confidence score
rclass_index = 4;
free_confidence_score = allScores(:,:,rclass_index);

% Display the free space confidence.
figure
imagesc(free_confidence_score)
title('Confidence Free Space')

% Resizing the image to that of CamVid sensor.
dimensions_image = sensor.Intrinsics.ImageSize;
I = imresize(I,dimensions_image);
free_confidence_score = imresize(free_confidence_score,dimensions_image);

% BirdsEyeView transformation
BirdsEyeViewImage = transformImage(birdsEyeConfig,I);
BirdsEyeViewFreeSpace = transformImage(birdsEyeConfig,free_confidence_score);

figure
imshow(BirdsEyeViewImage)
figure
imagesc(BirdsEyeViewFreeSpace)
title('Free Space Confidence')

% Defining the occupancy grid cells
X_grid = distAheadOfSensor;
Y_grid = 2 * spaceToOneSide;
cell_size = 0.25;

% Generate the occupancy grid
occupancyGrid = createOccupancyGridFromFreeSpaceEstimate(...
    BirdsEyeViewFreeSpace, birdsEyeConfig, X_grid, Y_grid, cell_size);

% Creating the BirdsEye plot.
birds_eye_plot = birdsEyePlot('XLimits',[0 distAheadOfSensor],'YLimits', [-5
5]);
hold on
[cell_count_Y,cell_count_X] = size(occupancyGrid);
X = linspace(0, X_grid, cell_count_X);
Y = linspace(-Y_grid/2, Y_grid/2, cell_count_Y);
% adding occ. grid
h = pcolor(X,Y,occupancyGrid);
title('BEV_Occupancy_Grid (probability)')
colorbar
delete(legend)
%Removing grid markers
h.FaceAlpha = 0.5;
h.LineStyle = 'none';

% Adding coverage area
cover_area_plot = coverageAreaPlotter(birds_eye_plot, 'DisplayName', 'Coverage
Area');

mountPosition = [0 0];
range = 15;
%orientation = 0;
%field_of_view = 60;
```

```
%plotCoverageArea(cover_area_plot, mountPosition, range, orientation,
field_of_view);
hold off

% Making cost map using the occupancy grid
cost_map = vehicleCostmap(flipud(occupancyGrid), ...
    'CellSize',cell_size, ...
    'MapLocation',[0,-spaceToOneSide]);
cost_map.CollisionChecker.InflationRadius = 0;

figure
plot(cost_map,'Inflation','off')
colormap(parula)
colorbar
title('Vehicle Costmap')

% Orientation correction in vehicle cordinates
% X axis in the direction of motion and Y axis pointing to left
view(gca,-90,90)

% Location points in world coords
input_points = [
    5 0.375
    4 0.375
    3 0.375
    4 2
    2 1.7];

% boolean check if occupied
isOccupied = checkOccupied(cost_map,input_points);

% Group the free and occupied locations
points_occupied = input_points(isOccupied,:);
points_free = input_points(~isOccupied,:);

% Plot the points on the cost_map.
hold on
markerSize = 100;
% free with green
scatter(points_free(:,1),points_free(:,2),markerSize,'g','filled')
% occupied with red
scatter(points_occupied(:,1),points_occupied(:,2),markerSize,'r','filled');
legend(["Free" "Occupied"])
hold off

%--------------------------FUNCTIONS---------------------------------

function occupancyGrid = createOccupancyGridFromFreeSpaceEstimate(...
    BirdsEyeViewFreeSpace,birdsEyeConfig,X_grid,Y_grid,cell_size)

% Total count of grid cells
cell_count_X = ceil(X_grid / cell_size);
cell_count_Y = ceil(Y_grid / cell_size);

% Defining the edges of every cell
X_edges = linspace(0,X_grid,cell_count_X);
Y_edges = linspace(-Y_grid/2,Y_grid/2,cell_count_Y);

% Specifying sample points for finding step size.
```

```
% Use these steps to cover entire cell area at the particular resolution

% Sampling 20 points from every grid cell
% More points,more computation
sample_points_count = 20;

X_step_size = (X_edges(2)-X_edges(1)) / (sample_points_count-1);
Y_step_size = (Y_edges(2)-Y_edges(1)) / (sample_points_count-1);

% Slide the set of points across both x and y direction of the cells.

% Use griddedInterpolant for finding occupancy probability.
probability_occupancy = 1 - BirdsEyeViewFreeSpace;
Size = size(probability_occupancy);
[y,x] = ndgrid(1:Size(1),1:Size(2));
GI = griddedInterpolant(y,x,probability_occupancy);

% Initializing the occupancy grid with zeros.
occupancyGrid = zeros(cell_count_Y*cell_count_X,1);

% Slide the set of points X_edges and Y_edges across both dimensions
for j = 1:sample_points_count

    % Incrementing sample points in the x-direction
    X  = X_edges + (j-1)*X_step_size;

    for k  = 1:sample_points_count

        % Increment in y-direction
        Y = Y_edges + (k-1)*Y_step_size;

        % Grid of sample points in BEV world coords
        [X_grid,Y_grid] = meshgrid(X,Y);

        % Transform from vehicle to image frame
        xy = vehicleToImage(birdsEyeConfig,[X_grid(:) Y_grid(:)]);

        % Clip points to lie within the frame dimensions
        xy = max(xy,1);
        xq = min(xy(:,1),Size(2));
        yq = min(xy(:,2),Size(1));

        % Sample occupancy probabilities using griddedInterpolant and keep
        % a running sum.
        occupancyGrid = occupancyGrid + GI(yq,xq);
    end
end

% Finding average occupancy probability.
occupancyGrid = occupancyGrid / sample_points_count^2;
occupancyGrid = reshape(occupancyGrid,cell_count_Y,cell_count_X);
end
```

Appendix H

CARLA LONGITUDNAL AND LATERAL CODE

```
                #!/usr/bin/env python3

"""
2D Controller Class to be used for the CARLA waypoint follower demo.
"""

import cutils
import numpy as np

# Initialize the class
class Controller2D(object):
    def __init__(self, waypoints):
        # Initialize all the parameters and set an
        # Initialization value
        self.vars                = cutils.CUtils()
        self._current_x          = 0
        self._current_y          = 0
        self._current_yaw        = 0
        self._current_speed      = 0
        self._desired_speed      = 0
        self._current_frame      = 0
        self._current_timestamp  = 0
        self._start_control_loop = False
        self._set_throttle       = 0
        self._set_brake          = 0
        self._set_steer          = 0
        self._waypoints          = waypoints
        self._conv_rad_to_steer  = 180.0 / 70.0 / np.pi
        self._pi                 = np.pi
        self._2pi                = 2.0 * np.pi


    # This Function Updates the latest state feedback from the CARLA Simulator
    # The function updates the x,y,yaw speed, timestamp, frame with the updates
    # Values
    def update_values(self, x, y, yaw, speed, timestamp, frame):
        self._current_x          = x
        self._current_y          = y
        self._current_yaw        = yaw
        self._current_speed      = speed
```

```
        self._current_timestamp = timestamp
        self._current_frame     = frame
        if self._current_frame:
            self._start_control_loop = True

    # This function calculates the desired speed for the controller
    # to track during the controller iteration
    def update_desired_speed(self):
        min_idx       = 0
        min_dist      = float("inf")
        desired_speed = 0
        # A loop is initialized so to find a minimum waypoint from the
        # List of the waypoints
        for i in range(len(self._waypoints)):
            dist = np.linalg.norm(np.array([
                    self._waypoints[i][0] - self._current_x,
                    self._waypoints[i][1] - self._current_y]))
            # the minimum distance vaiable is updated with the index
            # of the closest waypoint
            if dist < min_dist:
                min_dist = dist
                min_idx = i
        # Then the desired speed variable is updated with the speed
        # associated with that particular waypoint
        if min_idx < len(self._waypoints)-1:
            desired_speed = self._waypoints[min_idx][2]
        else:
            desired_speed = self._waypoints[-1][2]
        self._desired_speed = desired_speed


    #This function updates the current set of waypoints for the
    # coltroller to navigate
    def update_waypoints(self, new_waypoints):
        self._waypoints = new_waypoints


    # The simulator will recieve the output commands from this function
    # This functions returns the throttle, steer and brake values,
    # which will used by the simulation controller
    def get_commands(self):
        return self._set_throttle, self._set_steer, self._set_brake


    # This function sets the throttle command for the simulatior while bounds
    # the value to 0 and 1
    def set_throttle(self, input_throttle):
```

```
        throttle           = np.fmax(np.fmin(input_throttle, 1.0), 0.0)
        self._set_throttle = throttle


    # This function sets the steer command for the simulatior while bounds
    # the value to -1.22 rad and 1.22 rad
    def set_steer(self, input_steer_in_rad):

        input_steer = self._conv_rad_to_steer * input_steer_in_rad


        steer           = np.fmax(np.fmin(input_steer, 1.0), -1.0)
        self._set_steer = steer


    # This function sets the brake command for the simulatior while bounds
    # the value to 0 and -1
    def set_brake(self, input_brake):

        brake           = np.fmax(np.fmin(input_brake, 1.0), 0.0)
        self._set_brake = brake


# All the controllers are implemented inside this fucntion and all the above fun
ctions
# Are called inside this function
    def update_controls(self):

        x               = self._current_x
        y               = self._current_y
        yaw             = self._current_yaw
        v               = self._current_speed
        self.update_desired_speed()
        v_desired       = self._desired_speed
        t               = self._current_timestamp
        waypoints       = self._waypoints
        throttle_output = 0
        steer_output    = 0
        brake_output    = 0

        # persistent variables are defined here so that the
        # previous value can be used in a control loop
        self.vars.create_var('v_previous', 0.0)
        self.vars.create_var('t_previous', 0.0)
        self.vars.create_var('error_previous', 0.0)
        self.vars.create_var('integral_error_previous', 0.0)
```

```
self.vars.create_var('throttle_previous', 0.0)

# Skip the first frame to store previous values properly
if self._start_control_loop:



  #Longitudinal Control


    kp = 1      #Proportinal Gain
    ki = 0.2    #Integral Gain
    kd = 0.01   #Diffrential Gain

    # Throttle and Brake output
    throttle_output = 0
    brake_output    = 0

    # PID control
    st = t - self.vars.t_previous

    # Velocity error
    e_v = v_desired - v

    # Integral error
    inte_v = self.vars.integral_error_previous + e_v * st

    # Derivative error
    derivate = (e_v - self.vars.error_previous) / st

    # PID controller for acceleration
    acc = kp * e_v + ki * inte_v + kd * derivate

    # If acc value is positive assign it to the
    # Throttle variable
    if acc > 0:
        throttle_output = (np.tanh(acc) + 1)/2
        # throttle_output = max(0.0, min(1.0, throttle_output))
        if throttle_output - self.vars.throttle_previous > 0.1:
            throttle_output = self.vars.throttle_previous + 0.1
    else:
        # If the throttle value is negative make throttle 0 and brake 0
        throttle_output = 0
        brake_output = 0


    #Lateral Control
```

```
            # Change the steer output with the lateral controller.
            steer_output = 0

            # Stanley Controller used
            k_e = 0.3 # Defined the proportianl value
            slope = (waypoints[-1][1]-waypoints[0][1])/ (waypoints[-1][0]-
waypoints[0][0])
            a = -slope
            b = 1.0
            c = (slope*waypoints[0][0]) - waypoints[0][1]

            # Heading Error
            yaw_path = np.arctan2(waypoints[-1][1]-waypoints[0][1], waypoints[-
1][0]-waypoints[0][0])

            yaw_diff_heading = yaw_path - yaw
            if yaw_diff_heading > np.pi:
                yaw_diff_heading -= 2 * np.pi
            if yaw_diff_heading < - np.pi:
                yaw_diff_heading += 2 * np.pi

            # Cross Track Error
            current_xy = np.array([x, y])
            crosstrack_error = np.min(np.sum((current_xy -
 np.array(waypoints)[:, :2])**2, axis=1))
            yaw_cross_track = np.arctan2(y-waypoints[0][1], x-waypoints[0][0])
            yaw_path2ct = yaw_path - yaw_cross_track
            if yaw_path2ct > np.pi:
                yaw_path2ct -= 2 * np.pi
            if yaw_path2ct < - np.pi:
                yaw_path2ct += 2 * np.pi
            if yaw_path2ct > 0:
                crosstrack_error = abs(crosstrack_error)
            else:
                crosstrack_error = - abs(crosstrack_error)
            yaw_diff_crosstrack = np.arctan(k_e * crosstrack_error / (v))

            # Final Steering Equation
            steer_expect = yaw_diff_crosstrack + yaw_diff_heading
            if steer_expect > np.pi:
                steer_expect -= 2 * np.pi
            if steer_expect < - np.pi:
                steer_expect += 2 * np.pi
            # capping the steering values in between 1.22 rad and -1.22 rad
            steer_expect = min(1.22, steer_expect)
            steer_expect = max(-1.22, steer_expect)
```

```
        # Update the Steering angle
        steer_output = steer_expect

        # Set the values
        self.set_throttle(throttle_output)
        self.set_steer(steer_output)
        self.set_brake(brake_output)


    # Used to store old values

    self.vars.v_previous = v
    self.vars.throttle_previous = throttle_output
    self.vars.t_previous = t
    self.vars.error_previous = e_v
    self.vars.integral_error_previous = inte_v
```